Lúcio Sanchez Passos

Spectrum-Based Fault Diagnosis in Multi-Agent Systems

December, 2015

Spectrum-Based Fault Diagnosis in Multi-Agent Systems

Lúcio Sanchez Passos

A dissertation submitted to the
Faculty of Engineering, University of Porto
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Supervisor: Rosaldo J. F. Rossetti
Co-supervisor: Joaquim G. M. Mendes

GOVERNO DE PORTUGAL

GOVERNO FEDERAL BRASIL

FCT
Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

CAPES

POPH

QREN QUADRO DE REFERÊNCIA ESTRATÉGICO NACIONAL

UNIÃO EUROPEIA
Fundo Social Europeu

CIÊNCIA SEM FRONTEIRAS

The work presented in this thesis was supported by the *Fundação para a Ciência e a Tecnologia* (FCT) - grant SFRH/BD/66717/2009, by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) under the *Ciência Sem Fronteiras* programme - grant BEX 9382-13-5, by the Doctoral Programme in Informatics Engineering (ProDEI), by the Artificial Intelligence and Computer Science Laboratory (LIACC), and by the Institute of Science and Innovation in Mechanical and Industrial Engineering (INEGI).

Abstract

Spectrum-Based Fault Diagnosis in Multi-Agent Systems

Lúcio Sanchez Passos

Multi-Agent Systems (MASs) have been reshaping how complex problems are decomposed into distributed and decentralised solutions in different sectors. On the academic side, MASs open up many research opportunities to various existing fields by either experimentally exploring pre-established theories or bringing new abstraction paradigms that expand usage of current tools. On the industry side, the community stresses on the immense potential of MASs in real-world problems due to their pro-activeness, scalability, and reconfigurability through active entities with local perspective. However, the lack of reliability is pointed out as one of the reasons to the under deployment of such systems in real-world large-scale complex problems. The community therefore has particular interest in studying manners to ensure nominal MAS performance with maximum coverage.

A variety of methods and ideas have been proposed to ensure nominal performance of agent-based applications. This thesis provides a comprehensive survey of such techniques applied to improving reliability in MASs, starting by an overview of threats that can jeopardise the correct functioning of agents. These techniques usually focus on one part of the agent-oriented development life cycle. Hence, the relevant literature is deeply discussed regarding the topics of testing MASs, design validation based on simulation, and fault-tolerance in MASs. This review analyses aspects related to the designer perspective such as maturity level, MAS feature support, ease of use, and fault coverage in order to better ground our research.

Diagnosing unwanted behaviour in MASs is crucial to ascertain correct operation of agents. Current techniques assume *a priori* knowledge to identify unexpected behaviour. However, designing MAS' model is both error-prone and time consuming as it exponentially increases with the number of agents and their interactions. This thesis overcomes such dependency models by taking advantage of spectrum-based diagnosis techniques. We discuss the limitations of applying spectrum-based diagnosis in time persistent and autonomous entities such as agents and therefore propose the Extended Spectrum-based Fault Localization for MAS (ESFL-MAS). ESFL-MAS localises faulty agents in the system that may jeopardise

the overall performance through the computation of suspiciousness values.

Aiming at assessing the fundamental ESFL-MAS dependencies, we perform empirical evaluations using an exhaustive set of similarity coefficients. Experiments investigate how changing both the amount of collected data and the precision in detecting agent errors will affect ESFL-MAS' diagnostic quality. Assessments yield prominent results, giving a good prospect for the ESFL-MAS application to localise faulty agents. Results show that Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support coefficients give the best diagnostic accuracy (96.26% on average) in the boundaries of our experimental setup and are stable when varying either error detection precision and quantity of observations.

This thesis demonstrates that fault diagnosis in agent-based application can have high accuracy even relying on minimal information about the system, suggesting interesting challenges in the matter of detection and diagnosis processes being agnostic to programming language, agent architecture, and application domain.

Keywords: *Multi-Agent Systems, Spectrum-based Fault Localisation, Fault Diagnosis, Software Reliability.*

Resumo

Diagnóstico de Falhas Baseado em Espectro para Sistemas Multiagente

Lúcio Sanchez Passos

Os Sistemas Multiagente (SMAs) tem vindo a reformular como problemas complexos são decompostos em soluções distribuídas e decentralizadas em diferentes setores. No lado acadêmico, SMAs abrem diversas oportunidades de investigação para vários campos existentes através da exploração experimental de teorias pré-estabalecidas e trazendo novos paradigmas de abstração que podem expandir o uso de ferramentas atuais. No lado da indústria, a comunidade recalta o imenso potencial dos SMAs em problemas reais devido a sua proatividade, escalabilidade e reconfigurabilidade através de entidades ativas com perspectiva local. Todavia, a falta de confiabilidade é apontada como uma das razões para a inferior implantação desses sistemas em problemas complexos, reais e de larga escala. Dessa maneira, a comunidade tem um interesse em particular no estudo de maneiras de certificar a performance do SMA com uma certa abrangência.

Uma variedade de métodos e idéias tem sido propostas para assegurar a performance normal das aplicações baseadas em agentes. Esta tese provê uma revisão abrangente de tais técnicas aplicadas para a melhora de confiabilidade nos SMAs, a começar por uma visão global das ameaças que podem prejudicar o funcionamento correto dos agentes. Essas técnicas concentram-se geralmente em uma das partes do ciclo de desenvolvimento orientado à agentes. Portanto, a literatura relevante é discutida profundamente no que diz respeito aos tópicos de teste em SMAs, validação de desenho baseado em simulação e SMAs tolerantes à falhas. Este estudo analísa aspectos relacionados ao ponto de vista do desenvolvedor, assim como o nível de maturidade, o escopo de suporte em relação das características dos SMAs, a facilidade de uso e a cobertura dos tipos de falhas, tudo isso para fundamentar melhor esta tese.

O diagnóstico de comportamentos indesejádos em SMAs é crucial para garantir a operação correta dos agentes. Técnicas atuais assumem conhecimento *a priori* para identificar comportamentos inesperados. Entretanto, a construção do modelo do SMA é tanto propenso a erros quanto dispêndicioso de tempo, visto que esse cresce exponencialmente com o número de agentes e suas interações. Esta tese supera a necessidade de modelos aproveitando de técnicas de diagnóstico baseados em espectra. Nós discutimos as limitações da aplicação de diagnóstico baseado em espectra em entidades persistentes no tempo e autonômas, como os agentes, e assim propomos a Localização de Falhas baseada em Espectro Extendida para SMAs (*ESFL-MAS*). O *ESFL-MAS* localiza agentes faltosos no sistema, os quais possam prejudicar a performance total através do cálculo de valores de suspeita.

Visando avaliar as dependências fundamentais do *ESFL-MAS*, nós realizamos experimentos com uma lista completa de coeficiêntes de similaridade. Os experimentos investigam os efeitos da mudança da quantidade de dados coletados e da precisão da deteção de erros nos agentes na qualidade de diagnóstico do *ESFL-MAS*. A avaliação obteve efeitos proeminentes proporcionando um bom prospeto para a aplicação do *ESFL-MAS* na localização de agentes faltosos. Os resultados mostram que os coeficientes *Accuracy*, *Coverage*, *Jaccard*, *Laplace*, *Least Contradiction*, *Ochiai*, *Rogers and Tanimoto*, *Simple-Matching*, *Sorensen-Dice* e *Support* apresenta a melhor acurácia de diagnóstico (em média 96,26%) dentro das fronteiras do nosso sistema de teste, além disso são estáveis à variação tanto da precisão do detetor de erros quanto da quantidade de observações.

Esta tese demonstra que o diagnóstico de falhas em aplicações baseadas em agentes pode prover alta acurácia, mesmo confiando em um conjunto mínimo de informações sobre o sistema, revelando assim desafios interessantes no campo de processos de deteção e diagnóstico de falhas mais independentes da linguagem de programação, da arquitetura dos agentes e do domínio de aplicação.

Palavras-chave: *Sistemas Multiagentes, Localização de Falhas baseada em Espectro, Diagnóstico de Falhas, Confiabilidade em Software.*

In memoriam of my grandparents.

Vô Adolfo the Perspicacious.

Vó Maria the Sweet.

Vô José the Adventurer.

Vó Diná the Fighter.

Thank you for teaching me by example

the intangible things of life.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Dr. Rosaldo Rossetti for his support, which allowed me to both envision and pursue my ideas. In addition, I want to thank my co-supervisor Dr. Joaquim Gabriel for his generous support during these years.

I would like to thank the members of my dissertation committee: Prof. Dr. Eugénio Oliveira, Dr. Paulo Novais, Dr. Paulo Leitão, and Dr. Rui Abreu for providing me with invaluable comments and suggestions, which greatly contributed to improving this manuscript. In particular, I would like to give a special thanks to Dr. Rui Abreu for his collaboration and guidance.

It is paramount to acknowledge the government institutions and organizations that financially supported my research: the Fundação para a Ciência e a Tecnologia (FCT) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for providing the doctoral research grants, under the references SFRH/BD/66717/2009 and BEX 9382-13-5, respectively. I thank the Faculty of Engineering, University of Porto (FEUP), and the Artificial Intelligence and Computer Science Laboratory (LIACC) for the partial funding and support to this work.

I have been extremely lucky to cross paths with so many interesting people who somehow contributed to my personal growth. I would take this opportunity to thank them for all the words of comfort that motivated me to endure. Specially, I would like to show my appreciation to my dear friends Zafeiris Kokkinogenis, Leonardo A. Piedade, and Nuno Cardoso: thanks for our fruitful discussions, your time and assistance.

I am eternally grateful to my beloved Anabela N. Fernandes. She has been my lighthouse, from which I guided myself not to sink in the sea of stress, as well as my safe haven where I could take a breath, calm down, and be myself.

More importantly, I truly want to thank my mom Stela Maria Sanchez, my dad Ódinot R. dos Passos, and my sister Thaísa Sanchez Passos for being there. It seems to be pretty simple, but it is one of the most important things family can do. I apologise for worrying you and for being absent for so long.

Contents

List of Tables

List of Figures

List of Acronyms

AAA Adaptive Agent Architecture

ACL Agent Communication Language

ADELFE *Atelier de Développement de Logiciels à Fonctionnalité Emergente*

AI Artificial Intelligence

AOSE Agent-Oriented Software Engineering

ARCHON Architecture for Cooperative Heterogeneous On-line Systems

ARMOR Adaptive, Reconfigurable, and Mobile Object for Reliability

AUML Agent Unified Modeling Language

BDI Belief, Desire and Intention

CNet Contract Net

CNO Coordinated and Non-Organised

CO Coordinated and Organised

DaAgent Dependable Agent-based computing system

DAI Distributed Artificial Intelligence

DARX Dynamic Agent Replication eXtension

eCAT environment for the Continuous Agent Testing

EDP Error Detection Precision

ESFL-MAS Extended Spectrum-based Fault Localisation for Multi-Agent Systems

FACET Future ATM Concepts Evaluation Tool

FIPA Foundation for Intelligent Physical Agents

GOST Goal-Oriented Software Testing

IDE Integrated Development Environment

IDE-eli Integrated Development Environment for Electronic Institutions

IDK INGENIAS Development Kit

JADE Java Agent Development Framework

JAT Jade Agent Testing Framework

MAS Multi-Agent System

MAPC Multi-Agent Programming Contest

MASON Multi-Agent Simulator Of Neighborhoods

MASSIMO Multi-Agent System SIMulation framewOrk

MCMAS Model Checker for Multi-Agent Systems

NASA National Aeronautics and Space Administration

NCNO Non-Coordinated and Non-Organised

NCO Non-Coordinated and Organised

NP-hard Non-deterministic Polynomial-time hard

OCA Orbital Communications Adapter

OMAS Open Multi-Agent System

OO Object-Oriented

PASSI Process for Agent Societies Specification and Implementation

PASSIM Process for Agent Specification, Simulation and Implementation

PDP Pick-up and Delivery Problem

PDT Prometheus Design Tool

RatKit Repeatable Automated Testing Toolkit for Agent-Based Modeling and Simulation

REPAST Recursive Porous Agent Simulation Toolkit

SACI Simple Agent Communication Infrastructure

SaGE *Système avancé de Gestion d'Exceptions*

SeSAm Shell for Simulated Agent Systems

SFL Spectrum-based Fault Localisation

SG-ARP Server Group based Agent Recovery Protocol

SMA *Sistema Multiagente*

TAOM4E Tool for Agent Oriented visual Modeling for the Eclipse platform

TFG Technical Forum Group

TuCSoN Tuple Centres Spread over the Network

UIOLTS Utility Input-Output Labeled Transition System

VEPR Virtual Electronic Patient Record

XP Extreme Programming

Chapter 1

Introduction

Multi-Agent Systems (MASs) have been proposed in early 1980's as a promising software paradigm for complex distributed systems. It derives from an Artificial Intelligence (AI) sub-field concerned with concurrency of multiple intelligent problem-solvers, named Distributed Artificial Intelligence (DAI) [Bond and Gasser, 1988]. According to Gasser [1987], MAS "is concerned with coordinated intelligent behaviour among a collection of (possibly pre-existing) autonomous intelligent 'agents:' how they can coordinate their knowledge, goals, skills, and plans jointly to take action or solve (possibly multiple, independent) problems." Since then, the autonomous agent concept has been broadly studied in diverse fields [Demazeau et al., 2015].

On the academic side, MASs open up many research opportunities by either experimentally exploring pre-established theories or bringing new abstraction paradigms that expand usage of current tools. Modelling of complex systems has greatly improved, by considering autonomous entities (as agents) and their interactions, bringing to light holistic and microscopic phenomena once hidden within differential equations. More concretely, MAS is the state-of-the-art metaphor to model extremely heterogeneous societies [Dignum et al., 2002], automatises gathering and processing of huge amount of data from various domains [Cao et al., 2009], and provides a testing ground to areas demanding large number of entities such as game theory [Parsons and Wooldridge, 2002].

On the industry side, the community stresses the immense potential of MASs in real-world problems due to their pro-activeness, scalability, and reconfigurability through active entities with local perspective [Parunak, 1996, Leitão et al., 2012, Bazzan and Klügl, 2013]. Several successful deployments take advantage of these features such as well-known examples: the ARCHON [Wittig, 1992], the NASA's OCA Mirroring System [Sierhuis et al., 2009], the VEPR system [Cruz-Correia et al., 2005, Vieira-Marques et al., 2006], the work of Carrera and his colleagues [Carrera et al., 2012], the OMAS platform [Tacla and Barthès, 2003], the FACET simulations [Agogino and Tumer, 2012], and the software package MASSIVE[1]. These cases only scratch the surface on the subject and further examples might be found in the literature [Parunak, 2000, Manvi and Venkataram, 2004, Pěchouček and Mařík, 2008, Isern et al., 2010, Leitão and Vrba, 2011, Smith and Korsmeyer, 2012].

[1] MASSIVE (Multiple Agent Simulation System in Virtual Environment) is a high-end computer animation and artificial intelligence software package which has the ability to create thousands of agents that all act as individuals as well as react to its surroundings, including other agents. These reactions affect the agent's behaviour changing how they act by controlling pre-recorded animation clips.

Despite these vast number of research achievements, the MAS community created a Technical Forum Group (TFG)[2] to discuss the judged under exploration of agent-based applications in industrial environments. This TFG sought to answer the question of "Why are not Multi-Agent Systems largely used in real complex (distributed) systems?" and then pointed out several issues that contributed to such under exploration. One of them was the "risk" of implementing agents that have never proved their value in large-scale problems [Mcburney and Omicini, 2008]. Such negative perspective comes from the industry's conservatism about using MASs (susceptible to emergent behaviour) without any central decision unit [Pěchouček and Mařík, 2008]. As consequence, most practitioners avoid to use MAS-based solutions to fully control their critical tasks. To overcome this issue, it is necessary to expand and enhance the MAS development life cycle in order to assure that, given the demand for increasing functionality and flexibility on a short time to market, the system is released with proper testing and behaviour validation as well as prepared to deal with unexpected situations on a sound and safe basis. Therefore, the solution is to design and implement a more reliable MAS.

In addition to the industry's lack of acceptance cause by the autonomic aspect of MASs, agents are essentially software systems developed by humans under time, cost, and resource constraints. These constraints together with the inherent MAS complexity, almost unavoidably, leads to a growing amount of defects (a.k.a bugs[3]) in such applications. Unfortunately, software defects have caused several catastrophic failures throughout history, some of them entailing huge financial losses and life-threatening consequences [Charette, 2005, Wong et al., 2010]. MAS community learned from these previous events and recognises that dependability[4] plays a paramount role in preventing such hazards in MAS deployments.

Many research efforts aim at increasing reliability in agent-based applications and they can be mainly classified in four branches: (1) testing, (2) simulation-based design validation, (3) fault tolerance, and (4) formal verification (including model checking). The former is an essential phase in any software development process; a testing technique to MASs must support their challenging features such as distribution, decentralisation, and changeability. Simulation is a valuable means for assessing MAS outcomes when exposed to various (sometimes improbable) scenarios. Additionally, a fault-tolerant MAS maintains the provided service consistent with the system specifications despite the presence of an activated faulty component, which can derive from a wide range of points of failure. The latter is perhaps the most studied branch given the formal roots of MAS conception; these approaches ascertain whether or not all possible executions of a given MAS satisfy the set of requirements.

[2] Technical Forum Groups were a series of research meetings held under AgentLink III (2004-2005) and focused on topics suggested by the community. Readers interested in further information about TFGs can refer to Mcburney and Omicini [2008]. More information in `http://www.agentlink.org`

[3] The term bug was regularly applied to problems in radar electronics during World War II. Still, the term was first used in the scope of computer science in 1946 by Admiral Grace Hopper to report a defect in Harvard Mark II computer; in fact, a technician pulled an actual insect out from the contacts of the computer's relay.

[4] According to Avižienis et al. [2004], dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable to the user(s). It encompasses: availability (readiness for correct service), reliability (continuity of correct service), safety (absence of catastrophic consequences), confidentiality (absence of unauthorized disclosure of information), integrity (absence of improper system alterations), and maintainability (ability to undergo modifications)

Although these branches are complementary on providing reliable MASs, one can perceive by studying the literature two distinctive manners upon which the community is trying to advance towards this purpose. On the one hand, formal verification follows a more certification-oriented perspective that exhaustively and automatically explore the system state space targeting at any model inconsistency by using sophisticated mathematical tools. Such approaches are computationally costly and time consuming even while employing a number of boundaries and complexity reduction techniques, curbing their use in daily software development activities. On the other hand, the other three branches (testing, simulation-based validation, and fault tolerance) follow a more software engineering direction as they are concerned with practical matters of implementing agent-based applications. Across these last three branches, there is a fundamental task that dictates the performance of each and every leaf techniques: *to properly identify and diagnose faults in MASs.*

When unexpected behaviour is observed, one (developers or monitoring systems) needs to identify the root cause(s) that makes the MAS deviate from its intended behaviour. This task (also known as fault localisation or fault diagnosis[5]) is the most time-intensive and expensive phase of the software development cycle [Hailpern and Santhanam, 2002]. Besides, the unique and specific characteristics of MASs, namely autonomy and social ability, give rise to software faults not observed in any other type of software, exhibiting collective and individual unanticipated behaviours.

A number of research efforts have been devoted to developing techniques and tools for fault localisation in agent-based applications. This thesis aims to contribute to advancing the state of the art in automatic fault diagnosis in MASs.

1.1 Fault Localisation in Multi-Agent Systems

The process of localising the fault(s) that led to symptoms (failures/errors) is named fault diagnosis, and has been an active research area in both "traditional" and agent-oriented software engineering. Initially, this process has been done by developers through manually inserting *print* statements in source code to generate additional localisation information. Symbolic debuggers are alternative techniques that provide additional features such as breakpoints, single stepping, and state modifying. These debuggers are included in many integrated development environments (IDEs) such as Eclipse (which can be used with any Java-based agent framework) and Jason interpreter (in which beliefs and plans of an agent can be observed). To ease the burden in developers, automatic fault diagnosis approaches yield the likely location(s) of fault(s), which is subsequently used either by the developer to focus the software debugging process, or as an input to automatic recovery mechanisms [Patterson et al., 2002, Sözer et al., 2010]. In the scope of fault diagnosis in MASs, depending on the type of knowledge that is demanded about internal structure and behaviours of agents or interaction patterns, the relevant approaches can be classified as (1) fault-based diagnosis or (2) model-based diagnosis [Lucas, 1998]. The former uses experts knowledge about the implemented functioning to produce a list of probable faults in the MAS, whereas the latter combines a model specification of the expected behaviour

[5]In this thesis, the terms fault diagnosis and fault localisation are used interchangeably.

or type of interaction with a set of observations to localise the faulty component(s).

Fault-based techniques use a tree of possible exceptions given by system designers and/or domain experts as basis to detect and diagnose any observed failure. For instance, this fault model might be the likely causes of a lack of communication between two agents, such as: connection loss, protocol inconsistencies, and so on. Techniques usually search on these pre-defined trees stored in databases. Well-known examples of such approaches are the work of Chia et al. [1998], Dellarocas and Klein [2000], Horling et al. [2001], Roos et al. [2004], Guardian by Miller and Tripathi [2004], and more recently Shah et al. [2009]; all of which are described in more detail later on. Albeit the above proposals suggest fault-based diagnosis as solution to agents, there is a drawback when using it; synergy among agents is unpredictable and depends quite much on specific run-time conditions that restrict the design of a solid cause-fault model for them.

Model-based (a.k.a. consistency-based) approaches to fault localisation use prior system knowledge, such as component behaviours and interconnections, to build a model of the correct system behaviour. An example of a consistency-based technique is the plan diagnosis introduced by Roos and Witteveen [2009], where a diagnosis is obtained by analysing the set of actions that lead to failure while underling its root causes. Examples of this research line are the work of Roos, Jonge, and Witteveen [Roos and Witteveen, 2009, Jonge et al., 2009], and Micalizio [2009, 2013]. Another example is the social diagnosis in a MAS, which concerns in determining coordination failures within a team of agents. This is achieved by monitoring interactions within the team and then modelling behaviour hierarchy while agents update their internal model of teammates. Well-known example of social monitoring and diagnosis are the work of Kaminka and his colleagues [Kaminka and Tambe, 1998, 2000, Kaminka et al., 2002] and the work of Kalech and his colleagues [Kalech et al., 2007, Kalech and Kaminka, 2007, Kalech, 2012]. Moreover, there are many other techniques that rely on some kind of model specification such as source code (see, e.g., the JAT framework by Coelho et al. [2007]), interaction patterns (see, e.g., ACLAnalyser tool by Serrano et al. [2012]), design schemes (see, e.g., the eCAT framework by Nguyen et al. [2012]), expected performance (see, e.g., the work of De Wolf et al. [2006]), and so forth. The aforementioned techniques demand for *a priori* knowledge about the MAS, and later on it is going to be analysed in detail how differently they use the model specification.

To complement this discussion, other techniques that use prior information about the MAS are those based on model checkers. Such techniques automatically verify whether the agent model (and sometimes the whole MAS model) meets a particular set of requirements or not. They usually ascertain whether an agent can reach a critical state that might cause a failure, or check whether there is a deadlock in the execution or race condition-free. The most relevant examples of model checking in MASs include the MCMAS toolkit [Lomuscio et al., 2009, Lomuscio and Michaliszyn, 2014, Lomuscio and Paquet, 2015, Lomuscio and Michaliszyn, 2015, Čermák et al., 2015] and the Model Checking Agent Programming Languages project [Fisher et al., 2013, Dennis et al., 2014, 2015].

1.2 Problem Statement

As explained in the previous section, model-based approaches take as input some kind of information about the system, using the designer knowledge on the system under analysis. Such approaches have strong theoretical foundations in terms of the logical theories [de Kleer and Williams, 1987, de Kleer et al., 1992, Struss and Dressler, 1992, Console and Torasso, 1991] and therefore they have high diagnostic performance by exploiting the information present in the model. This information can be properly built when designers fully understand the environment upon which agents act as well as agents' state space.

However, despite the successful use of model-based diagnosis in different domains inside and outside the MAS arena, state-of-the-art techniques have handicaps while applied in large systems as pointed by recent literature on the subject [Gupta et al., 2014]. Manually creating intended behaviour models may be as laborious and/or error-prone as building the system itself and thus approaches to automatically derive a model from implemented source code have been proposed [Mayer and Stumptner, 2007], including for MASs (see, e.g. [Lam and Barber, 2005b,a]). Complete model could not be fully obtained using such approaches; for instance, in a case study in Dutch industry, only 80% of its model could be automatically built from its source code [Reeven, 2011]. Furthermore, in the perspective of continuous evolving systems, a simple software upgrade would demand for new tuning and/or calibration of the built model, which enforces more responsibility upon designers.

Even more complexity is added when agent's autonomy, dynamic environment, and other MAS features are taken into account while building behavioural models. Harmonically merge and/or interconnect them so the system's function can be properly represented might be extremely time-intensive and error-prone, being even impossible for particular cases. Therefore, this thesis intends to address *the lack of techniques that rely on minimal* a priori *knowledge to diagnose unexpected behaviours in Multi-Agent Systems.*

Software engineering community strives to effectively model software of realistic size and complexity (such as real-world agent-based applications) [Gupta et al., 2014]. Driven by this fact, research proposals consider statistical-based approaches that do not rely on behavioural models, such as the Spectrum-based Fault Localisation (SFL).

SFL is a promising technique that does not rely on an explicit model of the system under analysis. It uses an abstraction over program traces, also known as program spectra [Reps et al., 1997, Harrold et al., 2000], collected at run-time, during software execution, to discover a correlation between components and observed failures. The technique tries to localise the faulty component by computing their suspiciousness values using a similarity coefficient as heuristic. Components are subsequently ranked with respect to their suspiciuosness to be the root cause of failures. SFL has yielded good diagnostic performance for software systems in different domains, such as embedded software [Abreu et al., 2009, Le et al., 2013], spreadsheets [Hofer et al., 2015], and concurrent applications [Koca et al., 2013].

Given these properties, SFL is particularly interesting for diagnosis faults in MASs because (1) it has the potential to scale well to large MASs with numerous agents and artifacts, (2) it may be suited to resource-constrained application of agents, and (3) it is transparent to

agent architecture, implementation framework, and agent-oriented development process, assuming a set of test suite is available. Still, the SFL diagnostic accuracy is inherently limited by the available data and the used heuristic and it considers only single faults.

Aiming at proposing a model-less diagnosis technique to pinpoint behavioural faults in MASs, in this thesis we study a spectrum-based approach to diagnose faults in MASs. By mapping concepts of agent-based applications, fault diagnosis might benefit from absence of any model specification while maintain high diagnostic accuracy. In this perspective, we draw the following hypothesis:

Spectrum-based fault localisation algorithms can be adapted to better cope with both the time persistence and the autonomy of multi-agent systems.

In order to demonstrate the aforementioned hypothesis, a major effort has been put on analysing the spectrum-based techniques from the viewpoint of its potential application to MASs. In particular, this thesis intends to pursue the following research questions:

1. *How to encode both the time persistence and the autonomy into spectrum-based algorithms as a way of diagnosing behavioural faults in MASs?*

   Although there have been some approaches that applied Spectrum-based Fault Localisation to concurrent components, these components lack the temporal persistence and pro-activeness on their decision-making inherent from the fundamental concept of agents. We therefore need to rethink the SFL elements in terms of its applicability to MAS context.

2. *Which similarity coefficients have the best performance given the particular features of agent-based applications?*

   An important element of SFL algorithms is the similarity coefficient. If the coefficient is poorly chosen given the nature of data, the diagnostic performance is impaired jeopardising the technique's main purpose of localising faulty agents. As there is an enormous list of them, we need to study how each of them impacts on the diagnostic performance.

3. *What is the minimal threshold for both the amount of available information and the precision of error detection in which SFL for MASs sustains good performance?*

   Practical application of spectrum-based diagnosis suffers from being dependent on both the available information and the precision of error detector. Therefore, it is important to consider variations on both of these dependencies in order to establish an optimal usability boundary for the proposed technique.

4. *Can an SFL-based approach maintains high performance while searching for a faulty agent in various types of organisation and different coordination levels?*

Agents might use different mechanisms to coordinate actions while pursuing their goals as well as can organise themselves to optimise tasks and/or resource usage under some circumstances. These characteristics are paramount within the MAS paradigm and therefore we must investigate the effect that different coordination strategies and MAS organisations may have on discovering a faulty agent.

1.3 Contributions

This thesis improves the state of the art in fault diagnosis in multi-agent systems domain. Its main contributions on the subject are as follows:

1. A small set of previous work in MAS fault diagnosis has focused on agent behavioural fault, which could jeopardise the system performance (see, e.g. [Nguyen et al., 2012, Çakirlar et al., 2015, Fortino et al., 2014, Cossentino et al., 2008]). In Chapter 2, we show that such approaches unfortunately assume extensive knowledge about the MAS, including agent's model and interactions, while detecting and diagnosis faults. As a result, we discuss the limitations of applying SFL as it is commonly used with block-hit spectra to such software entities.

2. We propose the Extended Spectrum-based Fault Localisation for Multi-Agent Systems (ESFL-MAS) to diagnose agent behavioural faults. MAS concepts have been mapped to SFL basic elements, being agents and their expected behaviour at given time mapped to components and transactions respectively, thus incorporating time within the spectrum. Diagnosis at the system level steers the proposed technique towards a performance-oriented direction, resulting in the so-called *performance spectrum*. At last, some data do not effectively contribute to localise the faulty agent given the intensive monitoring of agents; so we suggest MAS-Filter that increases diagnostic accuracy by excluding low-entropy rows in spectra.

3. The ESFL-MAS performance has fundamental dependencies on (1) the similarity coefficient, (2) the quantity of observations, and (3) the quality of error detectors. As SFL has not been applied to agent-based applications, empirical assessment is therefore essential to discover the set of similarity coefficients that yields the best performance to MAS context as there is no standard one for all applications [Yoo et al., 2014, Hofer et al., 2015, Le et al., 2013]. We perform a thorough study on ESFL-MAS using an exhaustive list of similarity coefficients to clearly understand the influence of both the amount of available MAS information and the precision of an error detector on the ESFL-MAS performance within the boundaries of our experimental setup

4. As a more practical contribution, we create an experimental setup that can provide different types of information about the MAS. Such setup is an instance of the Pickup and Delivery Problem [Savelsbergh and Sol, 1995] as (1) it is well-known and (2) it is a real-world representative problem. Problems of this kind are highly dynamic, and decisions have to be made under uncertainty and incomplete knowledge, which is paramount to assess ESFL-MAS. In a nutshell, the scenario consists of mobile vehi-

cles retrieving and delivering a set of items, based on the AgentSpeak implementation presented in the Multi-Agent Programming Contest. We improved this implementation from two perspectives: first, implemented variants of the original version in order to create different levels of organisation and coordination; second, we inserted both hand-seeded and mutation-based fault in the agents. This experimental setup is sufficiently solid and sound to be used by the community as an initial benchmark to fault diagnosis in MASs.

5. Finally, we survey the research efforts building a consultation guide within the scope of this thesis. It extensively covers techniques to improve MAS reliability (except those based on formal verification), including namely: agent-oriented testing, simulation-based design validation, and fault tolerance approaches to agent-based applications. We also provide a classification based on the agent development cycle that assist newcomers to understand state of the art in each branch. More importantly, we assess the reviewed literature focusing on the (presumably) most important aspect for designers/developers, such as: maturity level, MAS feature support, ease of use, and fault coverage.

1.4 Thesis Outline

The five contributions outlined in the previous section are described in terms of three chapters. Chapter 2 overviews threats that can jeopardize correct agent functioning and then surveys the current literature on testing in MASs, simulation-based design validation, and fault-tolerant MAS. It also analyses aspects of the reviewed literature related to the designer perspective such as maturity level, MAS feature support, ease of use, and fault coverage. Chapter 3 describes a light-weight, automatic debugging-based technique, coined ESFL-MAS, that shortens the diagnostic process, while only relying on minimal information about the system. Chapter 4 presents experimental evaluation by varying (1) the amount of information and (2) the precision of error detection mechanisms while determining the best heuristics for ESFL-MAS. Finally, in Chapter 5 we draw conclusions, present recommendations for future work, and identify research trends and avenues for reliable MASs.

1.5 Origin of Chapters

The following list gives an overview of the publications that originated each chapter:

Chapter 2 has been submitted in ACM Computing Surveys. An earlier version of the chapter appeared in the *Encyclopedia of Information Science and Technology, 3rd ed.*, as a book chapter [Passos et al., 2015c].

Chapter 3 and Chapter 4 has been accepted in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. An earlier version of this work appeared in the *Proceedings of the 24th AAAI International Joint Conference on Artificial Intelligence (IJCAI'15)* [Passos et al., 2015a], in the *Proceedings of the 25th International Workshop on Principles of Diagnosis (DX'14)* (Best Paper Award) [Passos et al., 2014], and in the *Proceedings of the 11th European Workshop on Multi-Agent Systems (EUMAS 2013)* [Passos et al., 2013b].

Chapter 2

Survey on Increasing Reliability in Multi-Agent Systems

Since the early 1980's, the advent of computer networks and the introduction of Multi-Agent Systems (MASs) have been reshaping how complex problems are decomposed into distributed and decentralised solutions in different sectors [Wooldridge, 2009]. MASs are concerned with coordinated behaviour among a collection of autonomous, distributed, and intelligent entities, called agents [Gasser, 1987]. Many applications are taking advantage of MAS inherent features, such as entities dispersion, parallel reasoning, ill-structureness, and complexity [Parunak, 1996, Stone and Veloso, 2000]. Differently from objects, agents exercise control over their own actions and might pro-actively interact with other entities to achieve a determined goal [Jennings and Wooldridge, 1995]. At the same time, MASs are software and as such should respect the software development life cycle aiming at ensuring from requirement fulfilment to system validation. On the one hand, such unique and specific characteristics of MASs give rise to software faults not observed in any other type of software, exhibiting collective and individual unanticipated behaviours. On the other hand, they have encouraged researchers to endeavour in the development of more reliable real-world agent-based applications.

At the corporate level, techniques to ensure reliable MASs have become a fundamental component to agent-oriented software engineering methods. As the Technical Forum Group showed, the industry's conservatism regarding MAS emergent behaviour without any central decision unit is one of the issues that constrains the extensive deployment of MASs in real complex (distributed) domains [Pěchouček and Mařík, 2008]. Fortunately, there are no registered cases in which MAS failures have led to hazards or huge financial losses; however, unfortunately, there are such examples in software history [Charette, 2005, Wong et al., 2010]. Hence, reliable MASs from components to system perspectives have become of particular interest to researches and practitioners. A designer/developer or software company which is able to provide a tool that ensures nominal MAS performance with a certain coverage can help to improve acceptance of agent-based applications for solving complex problems.

In this chapter, we survey the various techniques used to increase reliability of every agent-based application levels. The more limited field of fault tolerance for mobile agents (in which agents are able to safely migrate within a network and continue execution even

is the presence of failure) is not discussed though[1], mainly because it involves a set of watchdogs and check-pointing procedures that are simply not applicable to non-mobile agents. Furthermore, although the certification sought by both model-checking and formal verification indeed contributes for more reliable MASs[2], this survey concentrates on testing and fault tolerance techniques inherent to the agent-oriented development life cycle. There are existing surveys on testing techniques for MASs [Nguyen et al., 2011], agent-based modelling and simulation [Fortino and North, 2013, Michel et al., 2009], and fault tolerance mechanisms [Passos et al., 2015c]. However, none of these do the profound discussion we intend to present in this survey.

The primary contributions of this chapter are as follows.

1. *Coverage.* The survey gathers publication efforts, covering techniques to improve MAS reliability with the aforementioned scope from its early origins to the state of the art. It includes testing and simulation-based new approaches and updates from the past recent years that were not discussed in the previous surveys. Moreover, this chapter extensively details research efforts of fault tolerance designed to suit agent-based applications.

2. *Classification and assessment.* The classification of approaches based on the development life cycle allows us to identify gaps in the literature, indicating possible MAS development phases that could benefit from further research and/or application of existing mechanisms. Similarly, the analysis of surveyed publications from the designer's/developer's point of view allows the community to identify some aspects that have yet to receive significant attention so such techniques are adopted by practitioners.

The remainder of the chapter is structured as follows: Section 2.1 provides an overview of fault-related concepts, with emphasis on how these concepts need to be changed to encompass particular MAS features; Section 2.2 outlines a taxonomy to approaches aiming at ensuring nominal MAS service; Section 2.3 describes various methods that are used to test and debug different levels of agent-based applications; Section 2.4 explains design validation mechanisms that benefit from simulation tools; Section 2.5 examines several advances in the area of fault-tolerant MAS; Section 2.6 analyses the reviewed literature mainly focusing on adoption aspects seen by designers; and Section 2.7 summaries the survey.

[1] Readers interested in the broader picture of how fault tolerance techniques are applied in mobile agents can refer to Isong and Bekele [2013]. Mobile agent had also been extensively used as fault tolerance providers and Qu et al. [2005] introduce several of these approaches.

[2] Readers seeking for deeper information about formal verification and model checking for MAS can consult Dennis et al. [2012] and Bordini et al. [2004]

2.1 Threats to Multi-Agent Systems

Before presenting and discussing the advances in reliable MASs, it is essential to explain the general terminology adopted across several domains. A *system* is an entity (which can be holonic[3]) that interacts with an external environment. It originates three major concepts: *function*, *behaviour*, and *service*. The former is what the system is intended to do. The second is what the system does to implement its function. And the latter is the perceived system behaviour. Chiefly, the correct behaviour is delivered when the service meets the system function.

When the delivered service deviates from the specification (i.e., the correct service), a *failure* event had occurred [Lee and Anderson, 1990, Avižienis et al., 2004]. This deviation may have different degrees of severity (a.k.a. failure modes). An *error* is a system state which is liable to lead to a failure. It is important to note that many errors do not reach the system's external state causing a failure. A *fault* is the adjudged or hypothesised cause of an error. It is *active* when it effectively causes an error, otherwise it is *dormant*. Faults might also be *transient* (those that cause temporary malfunction), or *permanent* (those in which, once it happens, the component fails permanently) [Laprie et al., 1992].

Fault, error, and failure are not isolated episodes. They have a deep connection that starts with the activation of either an internal previously dormant fault or an external fault. This now active fault causes an error in a given component state, which may propagate within the component (i.e., internal propagation) or from one component to another (i.e., external propagation). A failure effectively happens when the error propagates and reaches the service interface, provoking an incorrect delivered service. Figure 2.1 illustrates this fundamental chain of threats [Avižienis et al., 2004].

A known term also found in many publications regarding reliable MASs is *exception*. On its most general definition, "(exception) indicates that something unusual has occurred, and even this may be misleading" [Liskov and Snyder, 1979]. Hence, exception are anomalous conditions that change the normal software execution flow and requires handling processes. This term mainly refers to specialised programming language constructors or hardware mechanisms. It should be noticed that we use the terms *failure* and *exception* interchangeably throughout this thesis.

Now that basic terms to indicate general characteristics and failure-related events have been defined, we can discuss how an agent-oriented perspective tune each of them. The system term may be applied to different abstraction levels from the agent to high-level organisations (including hardware and/or legacy systems). Moreover, the vast majority of MAS research employs function, behaviour, and service terms interchangeably, meaning by them what the system does to implement its function. This might prevent a newcomer to fully understand the mechanisms to increase reliability and how to correctly explore them in different MASs.

[3] A *holon* is a self-similar structure that consists of sub-structures [Koestler, 1967]. In MASs, the vision of holons is usually related to the notion of composed agents. A holon encompasses both local and global, both individual and collective perspectives [Cossentino et al., 2010]. Readers interested in a thorough discussion on Holonic MASs can consult Fischer [1999] for a theoretical foundation and Leitão et al. [2012] for recent applications.

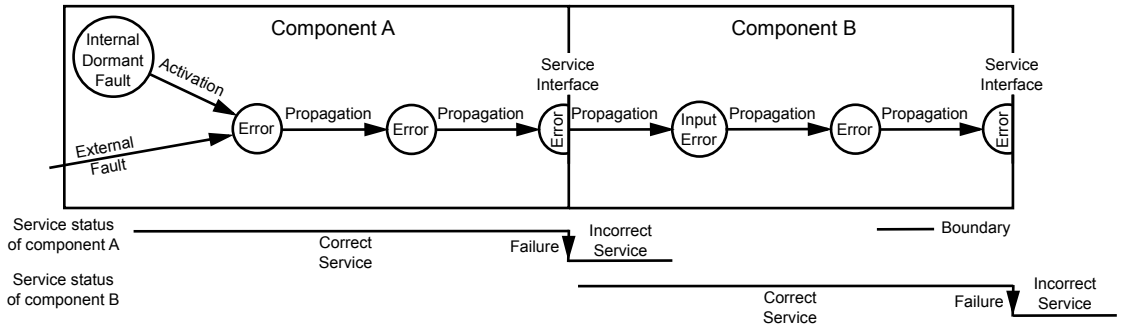


Figure 2.1: Error Propagation (Adapted from Avižienis et al. [2004])

Regarding the terms related to anomalous services, several endeavours extend the systemic view given by the failure definition to encompass the agent-oriented perspective. For instance, Tripathi and Miller [2001] define: "an exception is an event that is caused by a program using a language construct, or raised by the underlying virtual machine", which is also used by Souchon et al. [2004]. Complementarily, Xu and Deters [2004a] describe a fault as "the defect in the program that, when executed under particular conditions, causes a failure." Despite such endeavours, part of the MAS community states that these definitions do not completely comply with the agent paradigm [Platon et al., 2007], lacking allusion to issues of both autonomy and social interactions, as well as considering only non-social events as sources to unexpected situations.

Even though autonomy is an inherent feature of software agents, the aforementioned definitions disregard autonomous decisions by always considering an external source as origin of irregular events. Moreover, social interactions are a natural overlay of agent-based systems, where the "social fabric" may assist different system stages, e.g., to format its organization and/or exchange new methods to achieve optimal solution. Even in a fixed organization (i.e., defined by rules), informal associations can appear due to evolving interaction forms [Huberman and Hogg, 1996]. Aforementioned definitions also do not take social aspects into account as interaction violations focus only in low-level failures such as package loss. Thus, simply adapting definitions to cover agent faults restricts them to single threads and linear information/control flow; additionally, they do not cover the interactive nature of agent societies where faults encompass group of agents, themselves being multi-threaded software.

Seeking to overcome these constraints, some approaches define threats to MASs in rupture with traditional basis. Their main breakthrough is to consider faults as systemic matters. Klein et al. [2003] describe that "all (...) departures from the 'ideal' MAS behaviour can be called exceptions." Authors refer to "MAS behaviour" as a global conduct and thus, similarly to traditional approaches, a deviation from the "ideal" is a failure. The definition rises the term to a system level because MAS faults may spread throughout, affecting the whole system. Hence, the treatment dimension is not constrained to linear flow. Klein et al. [2003] focus on the subject of autonomy; however, disregard any social dynamism. The majority of works on dependable interaction among agents aims to improve protocol robustness, but no publication explicitly discusses an agent-oriented fault definition focusing on social aspects.

Furthermore, Platon et al. [2007] assert that the essential different between agent-related

and traditional failures is their source. Sources are not restricted to simple operational invocations because agents, owning autonomy, play a paramount role in determining execution status. So an event in either observable actions or system states should be comprehended in a broader sense whereas unexpected events are those that agents do not foreseen in a particular context. Authors propose five properties to agent failures: (1) the anomalous character of an event depends on how each agent interprets it; failures are broader than only programming or crash errors which implies in a neutral connotation for faults (Mallya and Singh [2005b] share this perspective and suggest distinguishing them between 'negative exceptions' from desirable ones, named 'opportunities'); (3) agents should have an internal knowledge to determine exceptional situations; (4) errors in MASs spread non-linearly, demanding for the asynchronous feature in exception-management mechanisms; and (5) an agent fault can exist without underlying programming exception, whereas programming exception implies an agent exception.

Another set of publications concerns less about what are failures in MASs and more on how classify them to better understand the state of the art. As seen above, a diversity of agent fault definitions may be found in the literature, in some cases complementary to each other and in other cases divergent. The same situation can be observed for the agent fault classification.

Faci et al. [2006] extend the generally accepted taxonomy of Powell et al. [1988], designed for general distributed applications, in which failures are classified based on the output produced by faulty components. This extension classify failures in four types. (1) The *crash failure* where the component stopped producing output. (2) The *omission failure* where a component suddenly ceases to output results. (3) When a result is delivered after a required time frame, it is called the *timing failure*. (4) The *arbitrary (or Byzantine) failure* relates to the event of generating random output values. Many endevours in the MAS literature use sub-groups or similar classifications to the above. Albeit this taxonomy encompasses various aspects of failures, it does not embody neither autonomy nor social interactions.

Similarly, Potiron et al. [2008] propose an extension of Avižienis et al. [2004]'s taxonomy, aiming to include autonomy as a new branch in the original taxonomy. New fault classes combine autonomy as an attribute to "phase of creation or occurrence" of faults. Authors also assemble faults into groups according to their origin. They divide behavioural faults in (1) development faults that are internal and (2) interaction faults that are external; both can yield catastrophic failure in the agent. This multi-level approach improves the accuracy of classifying a fault and goes further in this discussion, however it is complex for practical use.

In a simpler approach, Mellouli et al. [2004]'s taxonomy takes into consideration two aspects: the communication state and agent capability of performing actions. According to authors, if the communication is down, the agent may be fully capable of performing its tasks, partially capable, or incapable. Contrarily, if communication is up, the agent can only be partially capable or incapable of performing its tasks. This taxonomy neglects the real origin of the failure and concentrates on determining the functionality level of an agent.

Wagner [2000] presents a taxonomy for social order problems induced by software agents;

he applies an econometric theory, called liberal order, to the agent metaphor. According to this taxonomy, faults in agents can be: (1) *program bugs*, errors in the source code that go undetected through system testing; (2) *unforeseen states*, omission flaws in design/implementation; (3) *processor faults*, system or resources crash; (4) *communication faults*, various failures of communication links; (5) *emerging unwanted behaviour*, a system behaves differently from expected, which may be beneficial or harmful.

2.2 Overview of Approaches to Increase Reliability of MAS

Nearly all review and work publications in the area offer their own classification of the various approaches that have been developed for increasing reliability of agent-based systems. As this survey deals with complementary parts of the MAS life cycle, we have taken the classification of Moreno et al. [2009] as a starting point, with five levels of testing activities. Without loss of generality, we alter the Platon et al. [2008]'s taxonomy by combining the first groups in a larger class (because both of them basically rely on sentinel-like entities), excluding the stigmergic systems branch (as we consider that most of this research relates to swarm intelligence, which has been extensively surveyed by Qin et al. [2014]), and add a branch regarding replication-based architectures (due to the increasing popularity of these methods). We then merge these two modified taxonomies resulting in the classification that encompasses the scope of this chapter:

- *Testing in MASs* (*unit level*, *agent level*, *integration level*, and *testing frameworks*) works, which evaluate different MAS operational levels through predetermined cases during design phase, and provide tools to designers/programmers to better understand both behaviours and information flow during the implementation phase.

- *Simulation-based design validation* methods, which understand the system behaviour and/or assess various implemented strategies facing operational situations by modelling a limited set of both environmental features and dynamics.

- *Fault-Tolerant MASs* (*on-line monitoring and error detection*, *exception handling systems*, *sentinel-based architectures*, and *replication-based architectures*) approaches, which identify anomalous event origins (internal and/or external to agents), attempt to maintain minimal and consistent system services, and then recover from the faulty state aiming to return to nominal service.

Finally, we should mention that many of the testing approaches considered in the literature are components of frameworks, combining mechanisms from two or more levels. These frameworks will be described as a whole so the reader can perceive their progresses. We illustrate the proposed taxonomy in Figure 2.2. The main techniques will be reviewed in Sections 2.3 - 2.5.

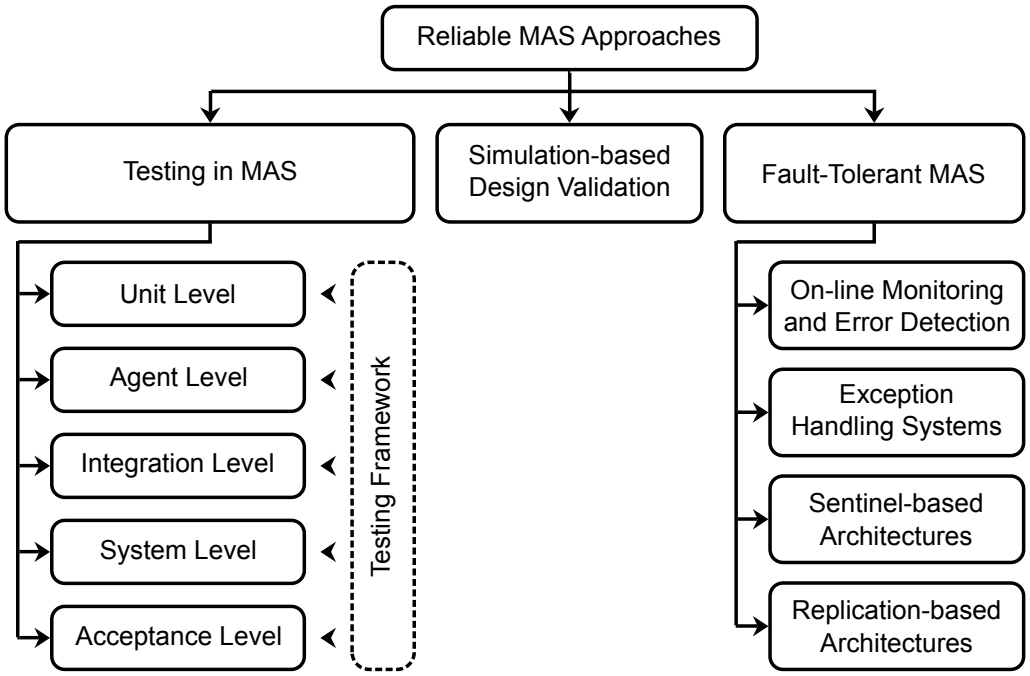


Figure 2.2: A taxonomy of approaches that intend to increase reliability of MASs with its life cycle. The main techniques are reviewed in Section 2.3 - 2.5.

2.3 Testing in MAS

Testing is a paramount activity in the software development life cycle that is dedicated to evaluate software quality, improve it by detecting implementation flaws, as well as assess the correspondence between delivered services and requirements. *Testing in MASs* is especially challenging given their distribution, decentralised decision-making, lack of structure, and so forth. As a result, MASs demand for new set of testing techniques that can deal with their unique features while ensuring highly reliable MAS with minimal costs and time spent. MAS body of knowledge spans techniques to test various system levels, being these: *unit*, *agent*, *integration*, *system*, and *acceptance*. Seeking clearness and balance, we present together unit and agent levels and separate integration level approaches. Additionally, we group works related to multi-level *testing frameworks* so the reader can easily follow their evolution.

2.3.1 Unit and Agent Levels

Agents are internally composed by modules with specific functions depending on their architecture. These modules may be a set of goals and plans, a knowledge base, several varieties of reasoning algorithms, and so forth, down to code methods and blocks. *Unit testing* makes sure that each of these aforementioned components independently delivers the expected output. Well-known unit testing techniques for sequential and Object-Oriented (OO) programming can be used in MASs to a certain extent.

Knublauch [2002] focuses on the Extreme Programming (XP) application, an agile methodology, as an alternative for waterfall-based Agent-Oriented Software Engineering (AOSE) methods. This work suggests the "agentification" of the XP process model. XP prescribes unit tests to assess each software module within an agent. These tests are based on the JUnit[4] test framework. The author provides an interesting application of agile methodology to MASs; yet, concerning the testing phase, this work directly applies OO methods (using JUnit) while it does not propose any solution to simplify generation and implementation of agent-oriented testing.

The next step is to increase granularity level and examine different modules within an agent. *Agent testing* also concerns with examining agent's behaviour given a set of environmental constrains while it tries to fulfil its goals. For instance, Coelho et al. [2006] aim to test agents individually by relying on Mock agents to guide both test case design and implementation. Both monitor and control of agents' asynchronous execution uses aspect-oriented mechanisms. Finally, they develop JAT, a test case generation framework [Coelho et al., 2007], which extends JUnit testing features to be applied on top of JADE[5]. Although this approach assures that tested agent complies with FIPA[6] specification, it does not assess any behavioural aspect of agent.

Lam and Barber [2004, 2005b] reverse engineering the agent implementation by mapping the model in terms of agent concepts. More specifically, they present the Tracing Method and the Tracer tool that compare actual behaviours of the implemented agent to their specifications. The method gathers dynamic run-time information and creates an agent behavioural model correlating beliefs, goals, and intentions. Then, using the Tracer tool, one can determine the implementation correctness and identify root causes of undesired behaviours. Authors complement the tool by adding model checking features [Lam and Barber, 2005a]. This research demonstrates the potential of gathering run-time information to create models and guide debugging for agent-based systems. However, authors do not go further and complement the Trace tool with automated monitoring and diagnosis mechanisms.

Pardo et al. [2010] propose a formal framework to specify e-commerce agents, called Utility Input-Output Labeled Transition System (UIOLTS) that embodies users' preferences in their states. Authors also introduce a conformance testing methodology composed by one active and other passive phases [Núñez et al., 2005]. In the former, a trace of the agent under operation given a certain stimulus is verified against specifications to discover faults. In the latter, the agent is observed without any stimulus and the trace of again compared to the specification. This approach is robust within the e-commerce domain, however its techniques are hardly extensible to other domains.

[4] `http://www.junit.org`.
[5] Java Agent Development Framework - `http://www.jade.tilab.com`.
[6] Foundation for Intelligent Physical Agents - `http://www.fipa.org`.

2.3.2 Integration Level

Two of the MAS paradigm's pillars are distribution and decentralisation, therefore interactions play a paramount role on such applications. *Integration testing* has the responsibility of ensuring that both communication (including protocols) and formation of any type of organisation do not provoke failures in MASs. Additionally, these techniques should observe emergent properties, collective behaviours, and diagnose any flaws. As one can see, given the complexity in designing such mechanism for a whole spectrum of organizations, this is perhaps the most under-explored level of MAS testing.

Nevertheless, ACLAnalyser[7] [Botía et al., 2006] is a software tool for debugging MASs that communicate using FIPA-ACL protocols. It selectivity monitors interactions between agents and saves every message in a database. The data analysis is twofold: (1) at group level, protocols are formally verified, and (2) at society level, clustering techniques discover emergent global behaviours. Authors propose a post-mortem analysis to detect emergency of unacceptable collaboration between agent using data collected during test executions [Serrano et al., 2009]. Experiments show that the tool is scalable to a large number of interactions. Finally, ACLAnalyser is agnostic to programming languages as long as ACL is agents' standard communication protocol [Serrano and Botia, 2009, Serrano et al., 2012].

2.3.3 Testing Frameworks

As testing is a major issue since the AOSE beginning, there are many research groups that work envisioning rather medium to long term results. Their research efforts mature to multi-level *testing frameworks* that are usually integrated into some known agent-oriented design methodologies. SUnit [Tiryaki et al., 2007] is one example that allows automated verification of agent behaviour (developed using SEAGENT[8] [Dikenelli et al., 2006]) and interactions between them. It extended the JUnit testing framework. This tool evolved to SEAUnit [Ekinci et al., 2009, Çakirlar et al., 2009] where agent goals are considered the smallest testable units in MAS. A goal possesses a set of requirement that should be fulfiled in the goal unit test; in the integration test, goal requirements might involve more than one agent.

Similarly, INGENIAS Development Kit[9] (IDK) [Gómez-Sanz et al., 2008, Gómez-Sanz et al., 2009] presents efforts to completely cover testing and debugging activities. A model-driven approach supports the design of proper test suites. Tests can cover deployment code, interactions, and agent mental states, which include special parameters in the code generation phase. A test is realised as a JUnit testing extension. Despite the approach eases the test specification, an automatic generation would increase both the coverage and the soundness of these tests.

[7] http://sourceforge.net/projects/aclanalyser
[8] `http://semanticweb.org/wiki/SEAGENT_Platform.html`
[9] `http://ingenias.sourceforge.net/`

Prometheus Design Tool[10] (PDT) [Padgham and Winikoff, 2004] supports both testing and debugging features. These are complementary components that help designers to develop MASs with Prometheus [Poutakidis et al., 2009]. Both Poutakidis et al. [2003] and Padgham et al. [2005] describe the debugging component that uses design artifacts. The approach debugs agent interactions using protocol specified in AUML-2 [Padgham et al., 2007] by converting them to Preti nets. Monitoring aspects of plan selection is also covered by the approach. The testing component uses model-based approach to perform automated agent testing [Zhang et al., 2011]. Although the work illustrates the framework with Prometheus, authors claim that the proposed process is extensible to similar methodologies. Events, plans, and beliefs are tested in a specific manner [Zhang et al., 2009]. The testing process consists of: (1) establishing the sequence to test system modules; (2) developing test cases; (3) enhancing the agent's source code; and (4) executing test cases and analysing results. Padgham et al. [2013] discuss the completeness and soundness of generated test inputs. Authors extensively evaluate the approach, which is able to detect a wide range of faults.

The eCAT[11] (environment for the Continuous Agent Testing) [Nguyen et al., 2008] framework generates and automatically executes test case. These test case skeletons are created from goal diagrams of Tropos methodology, using TAOM4E[12]. Goal-Oriented Software Testing (GOST) [Nguyen et al., 2010] supports test case creation from goal-oriented requirements. The methodology uses the V-Model to embody the mutual relationships between construction and testing of agents. Moreover, the testing process is done in five levels: unit, agent, integration, system, and acceptance. eCAT contains an ontology-based test case generation to assess interaction among agents by exploiting ontologies for managing inputs [Nguyen et al., 2009]. This generation has two main components: (1) the Tester Agent stimulates potentially faulty behaviour through sending messages; (2) the Monitoring Agent observes agent's reaction to the stimulus and reports any behaviour inconsistency. Authors suggest the use of evolutionary testing to better cover test case contexts [Nguyen et al., 2012]. The approach executes an initial population of agents; then collects data and calculates the fitness, which is based on requirements. If the fitness improves, good individuals are selected to create a better MAS.

2.4 Simulation-Based Design Validation

A common feature of agent-based applications is that their verification and validation is not a simple task due to their virtually infinite outcomes. *Simulation-based design validation* is showing promising advances in addressing this issue wherein simulation is used as a means for assessing MAS functioning by abstracting important features into models [Michel et al., 2009]. Such models play a paramount role in MAS validation. If the set of features chosen is not appropriate for capturing the main properties of agents, environment, and legacy systems, simulation results are likely to be unreliable. For a very good introduction to the synergy between Modelling & Simulation and MAS, readers are referred to Michel et al. [2009]. Our discussion does not intend to be an exhaustive list of every simulation-oriented

[10] `http://sites.google.com/site/rmitagents/software/prometheusPDT`
[11] `http://selab.fbk.eu/dnguyen/ecat`
[12] Tool for Agent Oriented visual Modeling for the Eclipse platform - `http://selab.fbk.eu/taom`

AOSE methodology; rather it focuses on chronologically presenting those approaches that explicitly describe the validation process as well as include some recent publication not included in the Fortino and North [2013] review.

The IDE-eli (Integrated Development Environment for Electronic Institutions) [Sierra et al., 2004] set of tools supports the development of MAS electronic institutions. Aiming at preventing hazardous behaviours in critical scenarios, the institution verification is twofold: static and dynamic. The former checks the structural correctness of the specifications. The latter is done via simulation where several populations with different settings are run with SIMDEI (based on REPAST[13]). IDE-eli's lack of automation in the verification curbs its expensive use as designers must manually assess the simulation results by confronting them against the expected ones without any auxiliary tools.

De Wolf et al. [2006] propose a mechanism that combines agent-based simulation with statistical algorithms for designing the macroscopic behaviour of self-organising MASs. In this approach, designers define the simulation expected outcome as well as supply initial values of macroscopic variables. Simulation are initialised in parallel, executed during a given period, and then measured by a set of pre-established metrics. The analysis algorithm compares the simulation results with the excepted outcomes and tunes the next round of simulation, stopping when the desired MAS performance is achieved. Authors additionally suggest an engineering process to integrate this systemic approach for designing MASs. This is an ad-hoc approach and, as pointed out by the authors, it should be integrated with an AOSE methodology.

Gardelli et al. [2006] evaluate the simulation use to detect anomalous behaviours in self-organizing and open MASs, which might be caused by: (1) wrong design choices, (2) the need for tuning agents' parameters, and/or (3) bugs in the implementation. They devise an architecture based on TuCSoN[14] infrastructure focusing on coordination between artifacts and agents. The verification of abnormal events is done using π-calculus to deal with large-scale simulations and then global properties are validated by comparing the gathered results with the designed models. Experiments are rather preliminary and authors disregard the abnormal behaviour detection in follow-up work [Gardelli et al., 2008].

Bernon et al. [2007] develop a model for self-organising agents so designers can more easily find behavioural inconsistencies. This model is implemented under the SeSAm[15] (Shell for Simulated Agent Systems) platform and then included in ADELFE[16], an AOSE methodology dedicated to adaptive MASs. The proposed tool enables the identification of any event that mitigates cooperation among agents by running prototyped versions of the MAS. Designer has to manually improve behaviours, which is not a good solution for large-scale MASs.

PASSIM is a simulation-based process for developing MAS [Cossentino et al., 2008]. It enhances the PASSI (Process for Agent Societies Specification and Implementation) methodology with the Java-based discrete-event simulation tool MASSIMO (Multi-Agent System SIMulation framewOrk) [Cossentino, 2005]. The process allows functional validation of

[13] `http://repast.sourceforge.net`
[14] `http://code.google.com/p/tucson`
[15] `http://launchpad.net/sesam`
[16] *Atelier de Développement de Logiciels à Fonctionnalité Emergente*, in English Toolkit to Develop Software with Emergent Functionality - `http://www.irit.fr/ADELFE`

agents' behaviours through generation of event traces, agent interactions protocols, and global MAS behaviour. The validation of global behaviour is off-line and completely relies on designers, being an error-prone and laborious task.

Sudeikat and Renz [2009, 2011] present a systemic approach to validate MASs. The procedure starts with formulation of hypotheses about intended macroscopic behaviours of the evaluated MAS based on given designs. These hypotheses are validated using a set of proper simulation settings in order to induce specific behaviours. Simulation generates logs that are analysed using correlations to discover any causal relation between environmental stimulus and agent behaviour. A prototyped environment was conceived to automate the aforementioned process. Despite the automatic MAS validation, the approach is not conceived to support complex agents and designer must build the validation model.

ELDAMeth[17] is a simulation-based methodology that encompasses (1) a visual agent model for distributed systems and (2) a supporting development tool [Fortino and Russo, 2012, Fortino et al., 2014]. Iterative simulations produce results measured by performance indices, which are derived from functional and non-functional requirements. Testing phase assesses agent acting upon given predefined scenarios and compare simulation results against requirements. Another component, the JADE-based platform, might generate a new code to be tested or remodel the system when requirements are not met. Authors rely on designers to establish performance metrics, not suggesting any guidelines to set such metrics.

The RatKit[18] (Repeatable Automated Testing Toolkit for Agent-Based Modeling and Simulation) [Çakirlar et al., 2015] is a tool for agent-based simulation to conduct verification, validation, and testing of agents and their interactions. The approach demands for three levels of requirements: micro, meso, and macro. The first considers the model of agents and interactions. The second tests requirements of sub-societies. The latter evaluates the global behaviour that emerge from individual entities [Gürcan et al., 2011, Gürcan et al., 2013]. RatKit has been implemented for REPAST and MASON[19] using the JUnit to write automatic test classes.

2.5 Fault-Tolerant MAS

A *fault-tolerant MAS* has the ability to maintain its behaviour consistent with its specifications despite the presence of an activated faulty component (which can be from a source code block to interactions within a sub-group of agents). Although there is no general recipe to completely achieve fault tolerance, it passes through four phases: (1) error detection, (2) damage confinement, (3) error recovery, and (4) fault treatment and continued system service [Lee and Anderson, 1990]. *On-line execution monitoring and error detection* focus on the first phase and thus observe agents in order to identify unexpected functioning during operation. In contrast, *Agent exception handling platforms* may deal with the whole fault tolerance process. They are especially useful in treating source code- and commitment-related errors. Moreover, many of the fault tolerance approaches rely

[17] `http://eldameth.deis.unical.it`
[18] `http://code.google.com/p/ratkit`
[19] Multi-Agent Simulator Of Neighborhoods - `http://cs.gmu.edu/~eclab/projects/mason`

on external entities - named *sentinels* - to supply reliability to agents. More recently, *replication-based architectures* maintain updated agent copies to activate them when facing a failure event.

2.5.1 On-line Monitoring and Error Detection

On-line Monitoring the MAS execution and *detecting errors* at run-time are two necessary steps in any fault tolerance scheme as they trigger every other activity to recover from exceptional situations. The accuracy of detecting anomalies is directly related to the amount of monitored information. If monitoring has access to every aspect of the system, an error detection mechanism is likely to be precise; at the same time, distributed monitoring consumes many resources. Thus, these approaches have to maximise detection precision while avoiding high costs.

In this perspective, Kaminka and Tambe [1998] propose the Socially Attentive Monitoring to failure detection particularly designed to collaborative agent systems. They use an explicit teamwork model to establish the difference in beliefs among agents. Team members monitor each other and detect any anomalous behaviour. The approach assumes that the agent's perception is only incomplete thus erroneous agents cannot be treated. Posterior research efforts focus on improving the monitoring method leading to the Overseer system [Kaminka and Tambe, 2000, Kaminka et al., 2002]. The system monitors large distributed applications by utilizing knowledge of the relationships between agents. Its knowledge base is built upon overhearing communications between team-mates.

Alternately, Kalech et al. [2007], Kalech and Kaminka [2007] concern with social fault diagnosis by modelling agent behaviours and updating agents' internal beliefs about teammates. Authors tackle scalability issue by improving the querying system, which has shown good results over other strategies [Kalech and Kaminka, 2011]. Recently, Kalech [2012] use matrix-based representation of coordination among agents to localise the faulty agent after the failure had been detected. They study non-binary constraints and improve efficiency to large-scale teams. This set of works is robust and scalable in collaborative MASs; however, they focus on the coordination failures rather than on how these may influence the MAS overall performance.

Chia et al. [1998] investigate the lack of coordination even when agents have a complete resource model. They classify undesired behaviours in poaching and distractions. Poaching is an event in which an agent reserves a resource, therefore preventing another (more critical) agent from consuming that resource. Distraction occurs whenever an agent receives information that misorients its course of actions. The author deployed a simple but effective solution for both problems: agents communicate their goal ratings to top n needed services and coordination mechanisms were added to guarantee resource scheduling after assigning tasks. Both mechanisms do not ensure performance improvement due to the complex set of interactions,

Horling et al. [2000] assume that agents must be able to generate situation-specific plans when a fault event occurs. Thus, they propose a performance-oriented architecture built upon the agent's task structure and expected interactions. The diagnosis system consists of

a lightweight comparison to determine any deviation. It has a predefined set of diagnosis rules and, according to the fault characteristics, specific set of rules are triggered. An interesting point of this work is the evaluation made to examine the effects of detection and diagnosis sensibility in the overall efficiency. However, the recovery strategy focus solely on finding a new task path and not on treating the existing fault.

Some research efforts focus on the fault localisation in agent plans, Roos and Witteveen [2009] introduce the primary plan diagnosis as the self-analysis of an agent's plan and a further endeavour [Jonge et al., 2009] extended this view with the definition of secondary plan diagnosis. Root causes of the plan failure are underlined by the latter whereas the former isolates the faulty set of actions. Moreover, Micalizio [2009] suggests an extended action model that might be used to identify and recover plans under partial observable situations. Recent efforts [Micalizio, 2013, Micalizio and Torasso, 2014] merge both model-based plan diagnosis and recovery schemes, ensuring a backup plan whenever possible. These techniques rely on prior agent plan model (that constraint their use in system testing) and mainly assume collaborative and non-malicious MASs.

2.5.2 Exception Handling Systems

Exception handing is a process inherited by MASs from software programming and hardware design. Handlers, by usually knowing the conditions that the exception occurred, can (depending on the error severity) either change the execution flow or resume the execution sending a report to designers. In the MAS context, exception management encompasses higher level concerns such as semantic inconsistencies and commitment violation.

Rustogi et al. [1999]'s approach handles semantic exception in a team of agents covering from the specification to execution of a MAS. In their view, an agent has two essential properties: interaction in a high level (commitments) and persistence. They develop a dependable interaction metamodel, which correlates the task definition with resource, capabilities, and so forth. In addition, they devise commitment patterns, enclosing the most common interactions. Any deviation on normal system flow is treated as a interaction problem only. This approach essentially addresses the issue of task disruption; however, the proposed treatment is to completely redesign the agent.

Xu and Deters [2004a] propose an event-based fault management that relies on agents' event reports and entities named event managers. Agents inform internal states and an event manager, which is equipped with event patterns, detects any faulty states. These managers correct agents via state-transition sequences whenever necessary. Authors apply the approach to a travel agency [Xu and Deters, 2004b] and in a third-party MAS [Xu and Deters, 2005]; concluding that (1) building event patterns is a laborious task, and (2) the approach has a scalability issue because, as every agent must report taken actions, the communication network is flooded with unnecessary messages.

SaGE (*Système avancè de Gestion d'Exceptions*, in English Advanced System of Exception Management) framework [Souchon et al., 2004, 2003] enhances the Java exception handling mechanisms to include particular features of agent-based systems. It combines exception handlers with a concerted exception mechanism, which gathers (and maintains history

of) fault information to handlers. SaGE complies with the agent paradigm preserving its autonomy and internal states. However, it does not support heterogeneous and open MASs as it assumes agents are always benevolent. Nevertheless, the approach suggests a set of novel techniques for MAS exception handling, namely handler search propagation and the concerted exceptions.

Platon et al. [2008] propose a multi-agent architecture that embeds exception management facilities for MAS deployment. It uses a twofold perspective. First, the exception-ready agent model extends beyond a basic agent architecture (composed by internal mechanisms and representations, actuators, and sensors), and encompasses exception handling logic within agents. Second, the environment is tailored to support specific exception types (e.g. agent death). Such functionality only notifies agents about environmental changes respecting their autonomy. The proposal depends on designers to provide information about the application domain, including specific data types and expected services, so agents can report exceptions.

Mallya and Singh [2005b]'s approach incorporates fault tolerance in commitment protocols to set how agents interact in an (error-prone) open system. When an agent detects an event that does not follow the protocol specification, it signalises an exception. The mechanism distinguishes the signalised exception between expected and unexpected. Excepted exceptions occur frequently and their treatment can be incorporated within the system. Unexpected exceptions are not modelled and hence require to dynamically handle them. Mallya and Singh [2005a] suggest an exception database that contains specific handlers with proper protocols (similar to Klein and Dellarocas [1999]'s model). Main issues of this work are that handler selection and assembly processes have high computational complexity and, at the same time, it is mainly theoretical and lacks performance evaluations.

2.5.3 Sentinel-based Architectures

The next class of work, known as *sentinel-based architectures*, tries to endow MASs with fault tolerance features through external software entities: the sentinels. Each sentinel assists an agent or a group of agents by inspecting interactions and actions. Additionally, many of the sentinel-based approaches considered in the literature have specialised sentinels to detect and recover an agent. A major issue with sentinels is that they do not respect agent encapsulation and therefore agent autonomy, as they are able to access agent's internal modules. Despite this fact, sentinel-based approaches have been the most used architecture for fault tolerant MASs.

Hägg [1997] introduces the use of sentinels to handle faults in MASs. According to the author, a sentinel is an agent whose mission is to guard a specific function or to avoid some states in the society. Their sentinel applies a twofold strategy to expand its judgement boundaries (responsible for detecting exceptions) and thus early detecting and treating agent inconsistency. First, it internally assembles its own world model by monitoring communications and interacting with other agents. Second, through an item checkpointing mechanism, some parts of other agents' model (expressed as beliefs) are integrated into the sentinel's world model. Albeit sentinels have recognised benefits, they are error-prone

entities that restrict agent autonomy to reduce the MAS space state.

Klein and Dellarocas [1999] propose a domain-independent exception handling in agent-based systems. The system uses expert sentinels to either (1) detect or (2) treat faults. They achieve generic behaviour through predefined strategies, to detect delays, low performance, and interaction or plan failures in agents. Yet, agents must register their complete behavioural model, so experts map failure modes and know *a priori* the possible faults. Generic diagnosis and recovery strategies are stored in a knowledge base. Communication with experts, as argued by the authors, are easily combined into the agent architecture. The approach was assessed using a simple error-prone Contract Net (CNet) domain with only one induced fault and, within this experimental boundaries, yielded good results [Klein et al., 2003, Dellarocas and Klein, 2000].

Adaptive Agent Architecture (AAA) [Kumar et al., 2000] is a fault-tolerant brokered multi-agent system architecture that aims to improve the robustness of communication among agents. Brokers form up a team with a joint commitment to support any registered agent, allowing that brokers substitute each other whenever required. The system maintains a minimal number of brokers despite failures in a subset of them. Their results suggest that the brokerage layer is fault tolerant [Kumar and Cohen, 2000]; however, it is not scalable due to the implied overhead that depends on the system size. This approach focuses on the recovery of brokers rather than on agents themselves, and does not show any assessment of how it increases reliability and availability of functional agents.

DaAgent (Dependable Agent-based computing System) [Mishra and Huang, 2000] improves agent-based computing dependability, mainly for Internet applications. Similarly to sentinels, an agent watchdog ensures that agents are able to (reliably) reach their destinations [Mishra, 2001]. In the presence of a (communication or node) failure, the agent rolls back to a checkpointed state done by its watchdog. A paramount contribution of this sequence of efforts is the identification of guidelines for an agent fault-tolerant protocol. This discussion culminates in the SG-ARP (Server Group based Agent Recovery Protocol) implementation and evaluation [Mishra and Xie, 2003].

Chameleon [Kalbarczyk et al., 1999] is an adaptive infrastructure that allows different availability levels in a networked domain. It is built upon ARMORs (Adaptive, Reconfigurable, and Mobile Objects for Reliability), which embodies specialised agents [Iyer et al., 1997]. ARMOR can be (1) managers (including fault tolerance manager), (2) daemons, and (3) commons. The approach is designed to recover mainly from hardware faults. Extensive evaluation is carried out and, according to the authors, the overhead and response time is acceptable [Kalbarczyk et al., 1999]. The ARMOR middleware offers a high-dependability services to application and has been deployed in several fields [Kalbarczyk et al., 2005]. The concept of adapting the fault treatment technique according to the system needs is an appealing feature. Moreover, the system is not necessarily specific to MAS, therefore it neither diagnose nor treat every agent failure.

The Guardian model [Miller and Tripathi, 2004] extends sequential exception handling models to distributed (including agent-based) applications. It centres on sentinel, called guardian, that monitors interaction among agents and uses predefined models to detect and treat a fault. An agent can notify and send an exception to its guardian. They deploy the Guardian in the Ajanta mobile agent programming system [Tripathi and Miller, 2001]. The

Guardian model usage is limited to open MASs, given the agent benevolence assumption where agents must not hide an exception from its guardian.

2.5.4 Replication-based Architectures

Redundancy is a key concept for fault tolerance as it includes additional resources that will only be activated in the presence of failure. *Replication-based architectures* has borrowed this concept and applied it to the agent level. However, to simply copy the information of an agent before its failure and then pass it to a replica might not be enough to bring the system to error-free state. These architectures demand for check-pointing and replica creation strategies without affecting nominal services.

Fedoruk and Deters [2002] introduce the use of dynamic proxies to manage groups of agent replicas. They deal with key challenges of agent replication, such as: synthesis of agent interaction and outcomes, state synchronization, and read/write consistency. In the conceptual architecture, a replica group is composed by a given number of host agents, each equipped with a message proxy, a replica agent, and a group manager [Fedoruk and Deters, 2003]. Experiments done with FIPA-OS[20] reveal that the system was effective in improving multi-agent reliability and demonstrate the potential of redundancy in a fault-tolerant scheme.

DARX[21] [Guessoum et al., 2010] is an adaptive replication-based framework for increasing reliability of distributed applications, which provides schemes to ensure nominal system functioning agnostic to agent architectures. DARX architecture applies three strategies: (1) dynamically manage replica membership, (2) provide the infrastructure so replication groups are able to internally communicate, and (3) agent interactions (external to the replication group) are gathered by individual replica and transmitted in cases of recovery procedures. Moreover, authors integrate DARX with a multi-agent platform (named DIMA [Guessoum and Briot, 1999]) resulting in DimaX [Faci et al., 2006]. Several pieces of work [Guessoum et al., 2003, Marin et al., 2007, Almeida et al., 2008] were implemented as improvements to this framework; however, to describe them all it is outside the scope of this review. Some recent endeavours [Dony et al., 2008, 2011] aim to combine exception handling and agent replication by adding a new level to the architecture. In this level, authors have incorporated the SaGE framework as the exception handling tool. This approach is perhaps the most complete from the reliability point of view as it provides preventive and corrective solutions; however, its resource usage will always be doubled regardless any optimisation given that every agent must have at least one replica.

[20] `http://fipa-os.sourceforge.net`
[21] `http://pages.lip6.fr/Olivier.Marin/DARX.startup`

2.6 Analysis of Techniques Aiming at Reliable MAS

Different schemes, mechanisms, and architectures have been proposed aiming at ensuring nominal service of agent-based applications from the design phase with testing and simulation-based validation, to the operational phase with fault tolerance support. After classifying and describing the relevant literature in Sections 2.3 - 2.5, it is essential to analyse advances reported in the literature so as to identify tendencies and better understand paths chosen by researchers as they deal with faulty events.

A few publications in the literature have analysed approaches within the scope of this chapter. For instance, Nguyen et al. [2011] assess testing techniques in terms of their maturity level and classify them as usable, in progress, and concept. From the fault tolerance perspective, Xu and Deters [2005] make a broad multi-dimensional study that emphasizes five key features: (1) time of intervention of the mechanism, (2) roles of system and agent, (3) required information, (4) required changes, and (5) scope of support. Both of these studies highlight interesting aspects, however they disregard the fact that designers demand additional time to analyse and choose a proper solution.

Designers consider both usefulness and ease of use to be important factors while choosing or adopting information systems [Keil et al., 1995]. The former is related to "the degree to which a person believes that using a particular system would enhance his or her job performance." The latter is rather related to "the degree to which a person believes that using a particular system would be free from effort." We derive four adoption-related features from both usefulness and ease to use concepts as this chapter intends to assess the surveyed literature considering the viewpoint of the designer. Techniques and tools proposed by the community have different *maturity levels* as they might be either in progress or solely be an exploratory study. *MAS Feature Support* inspects every set of publication regarding the major MAS features. *Ease of Use* aims at discussing which modifications must be done and/or information must be offered so an approach can properly work. *Fault Coverage* uses Wagner [2000]'s fault classification presented in Section 2.1 to comprehend their scope.

2.6.1 Maturity Level

As we can see throughout this chapter, there are several proposals that try to ensure correct agent functioning by either avoiding failures or overcoming them. These proposals are not in the same level of development due to factors such as (1) the number of members involved in the project, (2) achievement regarding performance and/or publication results, and so forth. Therefore, designers must take into consideration the maturity level of the chosen approach as this may require more implementation to obtain the desired results. Maturity level can be: *usable* (these have published results, available material to experiments reproduction, and have been applied to medium to large projects), *in progress* (these have published results, available material to experiments reproduction, and have been assessed using small or academic projects), or *concept* (these have published results but are in the theoretical level) [Nguyen et al., 2011].

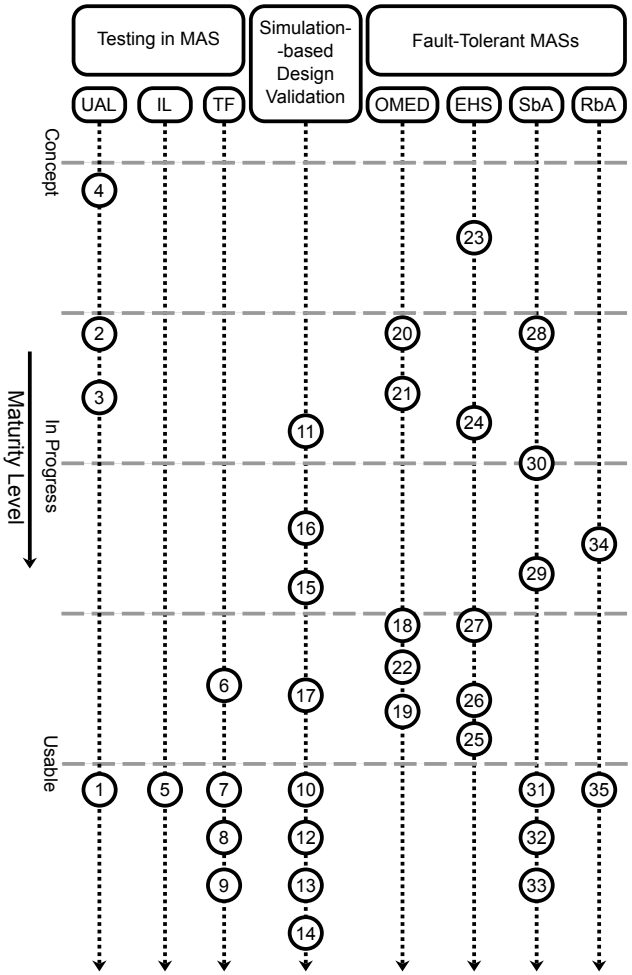
Table 2.1 shows the distribution of research efforts with respect to their maturity level. Those near the Concept line were minimally experimented; proposals near the In Progress line were experimented using toy problems and/or small applications. Research efforts above the Usable line had various medium-size assessments, but lack trials in more realistic/large problems; whereas those below are indeed usable within their scope of MAS feature support. It is important to highlight that, for the sake of clearness, we used only the more relevant publications in the legend.

One may notice that testing frameworks and simulation-based approaches are current hot topics to the community given the major number of relevant publications in the past recent years. Another interesting aspect is that these projects produce usable tools, which confirms the need for such readiness-orientation of the research on reliable MASs. Still, this high number of usable approaches may lead to the false sense that both of these branches have already developed "silver bullets" to guarantee the nominal agent service. Actually, these proposals use the strategy of restricting their focus to a particular type of MAS and consequently better defining usage boundaries; further discussion about this topic will be given in the next analysis.

Furthermore, although ACLAnalyser [Serrano et al., 2012] is the only technique for integration testing, it provides a robust API to support verification of ACL-based communication to such an extent that it was included in IDK [Gómez-Sanz et al., 2009] and might be integrated in any testing tool. This approach should be seriously considered to perform integration testing MASs relying on ACL-based interactions.

Concerns regarding fault tolerance in MASs have boosted publications between early to mid 2000's, but most of them were ad-hoc techniques because they used toy problems as experimental setup and since then no other means of validation was provided or was any of these works followed up. Nevertheless, these approaches laid the foundations to more complete and recent work such as Chameleon [Kalbarczyk et al., 2005] and SaGE [Souchon et al., 2004]. The former is more recently designated as ARMOR middleware and has been deployed to domains outside the MAS arena. The latter as mentioned in Section 2.5.4 was incorporated in the most robust fault tolerance architecture, namely DARX [Guessoum et al., 2010], to provide support to low-level exceptions. In conclusion, although exception handling systems for MASs had achieved great results in the last decade, sentinel-based and replication-based architectures better meet distribution and decentralisation aspects of agent-based applications with lower maintenance costs.

Table 2.1: The maturity level of the reviewed literature taking into account the amount of experiments. Note that only the most relevant publication of the respective project was selected to appear in the legend.



* UAL: Unit and Agent Levels; IL: Integration Level; TF: Testing Frameworks.

** OMED: On-line Monitoring and Error Detection; EHS: Exception Handling Systems; SbA: Sentinel-based Architectures; RbA: Replication-based Architecture.

Legend
(1) – [Knublauch, 2002]
(2) JAT [Coelho et al., 2007]
(3) – [Lam and Barber, 2005a]
(4) – [Pardo et al., 2010]
(5) ACLAnalyser [Serrano et al., 2012]
(6) SEAUnit [Çakirlar et al., 2009]
(7) IDK [Gómez-Sanz et al., 2009]
(8) PDT [Padgham et al., 2013]
(9) eCAT [Nguyen et al., 2012]
(10) IDE-eli [Sierra et al., 2004]
(11) – [Bernon et al., 2007]
(12) ELDAMeth [Fortino et al., 2014]
(13) – [De Wolf et al., 2006]
(14) PASSIM [Cossentino et al., 2008]
(15) RatKit [Çakirlar et al., 2015]
(16) – [Gardelli et al., 2008]
(17) – [Sudeikat and Renz, 2011]
(18) – [Kaminka et al., 2002]
(19) – [Kalech, 2012]
(20) – [Chia et al., 1998]
(21) – [Horling et al., 2000]
(22) – [Micalizio and Torasso, 2014]
(23) – [Rustogi et al., 1999]
(24) – [Xu and Deters, 2005]
(25) SaGE [Souchon et al., 2004]
(26) – [Platon et al., 2008]
(27) – [Mallya and Singh, 2005a]
(28) – [Hägg, 1997]
(29) – [Klein et al., 2003]
(30) AAA [Kumar et al., 2000]
(31) DaAgent [Mishra and Xie, 2003]
(32) Chameleon [Kalbarczyk et al., 2005]
(33) Guardian [Miller and Tripathi, 2004]
(34) – [Fedoruk and Deters, 2003]
(35) DARX [Guessoum et al., 2010]

2.6.2 MAS Feature Support

MAS field have been extensively investigated to solve complex problems present in several applications. As a result, a vast range of technologies and techniques have been envisioned to build such systems. There are four main MAS features that designers should observe while electing a validation, testing, and/or fault tolerance approach: (1) agent architecture, (2) type of organisation, (3) openness, and (4) heterogeneity. *Agent architecture* dictates how the abstract concept of agent is actually implemented in source code. This influences many aspects from possible information to be monitored to time responsiveness. *Type of organisation* defines the social fabric of agents and thus, in the presence of an error, it can help to discover the faulty component and prevent an overall failure by communication structures. Horling and Lesser [2004] extensively describe MAS organizational paradigms. *Openness* of a MAS regards its numbers of agents; if new agents can be created and/or enter the system, the MAS is open, otherwise, it is close. *Heterogeneity* relates to the uniformity of the agent architecture present in a MAS, being homogeneous when all agents have the same architecture and heterogeneous otherwise.

By the comparison showed in Table 2.2, it is important to note that:

1. Knublauch [2002] is the only testing proposal that can handle faults in every agent architecture, because it deals with low-level component within agents.

2. No approach concentrates purely on simple reflex (also called reactive) agents. A possible reason stems from the swarm intelligence field, which studies the usage of these type of agents to conceive emergent behaviour for solving complex problems. They state that reactive agents are inherently robust as a failure in a small set of them does not harm the system evolution.

3. Regarding those approaches that clearly declare the type of agent architecture/programming language, more than half uses JADE or Java to implement MASs; although they are generalist, JADE and Java demand for longer development time, which may affect the rapid prototyping philosophy that grows in the AOSE methods.

Complementing the discussion about boundaries of usable approaches of both testing frameworks and simulation-based validation, on the one hand, PDT [Padgham et al., 2013] and eCAT [Nguyen et al., 2012] support goal-oriented (specifically BDI) architectures. De Wolf et al. [2006] and IDE-eli [Sierra et al., 2004] are designed to assess only team-oriented MASs and electronic institutions respectively. On the other hand, IDK [Gómez-Sanz et al., 2009], ELDAMeth [Fortino et al., 2014], and PASSIM [Cossentino et al., 2008] comfort a wide range of agent architectures yet in a closed and heterogeneous MAS. Hence, designers/developers must know in advance the MAS features before choosing such tool.

We propose the following reflection to justify the predominance of work that deal solely with close MASs. Suppose that the proposal is able to improve its agent models over time, for homogeneous MASs, it can guarantee that in certain point in the future the identification and/or response to faulty events is going to be optimal. As there is only one kind of agent, it will be able to precisely model the agent functioning. Analogously to heterogeneity, when the MAS is close (new agents cannot enter), the system can also ensure

Table 2.2: Comparison of proposals with respect to the MAS Features Support. Note that only the most relevant publication of the respective project appear below.

Proposed Approach	AA	AF/PL	ToO	Op	He
Testing in MAS					
– [Knublauch, 2002]	SR, RS, Gb, Ub	1	Any	□	△
JAT [Coelho et al., 2007]	SR, RS, Ub	2	Any	□	△
– [Lam and Barber, 2005a]	Gb	1,5,8	Any	□	△
– [Pardo et al., 2010]	Ub	–	–	–	–
ACLAnalyser [Serrano et al., 2012]	SR, RS, Ub	2	Any	□	△
SEAUnit [Çakirlar et al., 2009]	RS, Gb	8	–	□	△
IDK [Gómez-Sanz et al., 2009]	SR, RS, Ub	2	Any	□	△
PDT [Padgham et al., 2013]	Gb	4	Any	□	△
eCAT [Nguyen et al., 2012]	Gb	2,3	Any	□	△
Simulation-based Design Validation					
IDE-eli [Sierra et al., 2004]	SR, RS, Gb	1	EI	▽	△
– [De Wolf et al., 2006]	SR, RS, Gb	–	T	□	◯
– [Gardelli et al., 2008]	SR, RS, Ub	7	C	□	◯
– [Bernon et al., 2007]	SR, RS	6	C	□	◯
PASSIM [Cossentino et al., 2008]	SR, RS, Gb, Ub	2	Any	□	△
– [Sudeikat and Renz, 2011]	Gb	3	Any	□	△
ELDAMeth [Fortino et al., 2014]	SR, RS, Gb, Ub	2	Any	□	△
RatKit [Çakirlar et al., 2015]	SR, RS, Ub	7	C	□	◯
Fault-Tolerant MAS					
– [Kaminka et al., 2002]	RS, Gb, Ub	–	T	□	◯
– Kalech [2012]	Gb, Ub	–	C	□	△
– [Chia et al., 1998]	RS, Gb, Ub	–	C	□	◯
– [Horling et al., 2000]	SR, Gb	–	C	□	◯
– Micalizio and Torasso [2014]	Gb, Ub	–	C	□	△
– [Rustogi et al., 1999]	SR, RS, Gb, Ub	–	Any	□	◯
– [Xu and Deters, 2005]	SR, Rs, Ub	2	Any	□	△
SaGE [Souchon et al., 2004]	RS, Gb, Ub	1	Any	□	◯
– [Platon et al., 2008]	RS, Gb, Ub	–	Any	▽	△
– [Mallya and Singh, 2005a]	RS, Gb, Ub	–	Cb	□	△
– [Hägg, 1997]	SR, RS	8	Any	▽	◯
– [Klein et al., 2003]	RS, Gb, Ub	–	Any	□	◯
AAA [Kumar et al., 2000]	SR, RS, Gb, Ub	–	C	▽	◯
DaAgent [Mishra and Xie, 2003]	SR, RS, Gb, Ub	–	Any	▽	△
Chameleon [Kalbarczyk et al., 2005]	SR, RS, Gb, Ub	–	Any	▽	△
Guardian [Miller and Tripathi, 2004]	SR, RS, Gb, Ub	–	Any	□	◯
– [Fedoruk and Deters, 2003]	SR, RS, Gb, Ub	–	Any	▽	△
DARX [Guessoum et al., 2010]	SR, RS, Gb, Ub	–	Any	□	△

* AA: Agent Architecture - SR: Simple Reflex; RS: Reflex with States; Gb: Goal-based; Ub: Utility-based.

** AF/PL: Agent Framework/Programming Language - 1: Java; 2: JADE; 3: JADEX; 4: JACK; 5: C/C++; 6: SeSAm; 7: REPAST; 8: Others.

*** ToO: Type of Organisation - EI: Electronic Institution; T: Team; C: Collaborative; Cb: Commitment-based.

**** Op: Openness - □: Close; ▽: Open.

***** He: Heterogeneity - ◯: Homogeneous; △: Heterogeneous.

that at some point in the future the fault tolerance scheme will have models for all possible agents. This latter process will certainly take longer than the former but theoretically it also leads to optimal solutions. However, when the MAS is open, there is no guarantee that the testing, validation, or fault tolerance will be able to even detect an anomalous event. Therefore, most of the efforts intends to address the reliability issue first to close MASs, envisioning further generalisation to open MASs.

Most of authors disregard the type of organisation because they assume that their approach is robust in all of them; however, their experimental setups do not cover every organisational structure. Nonetheless, a collaborative society opens up the possibility to use team-mates as comparison basis [Kaminka and Tambe, 1998] and, at same time, sets clear boundaries to possible treatments because only benevolent agents are considered. However, agents that either sends incorrect information or act to jeopardise another agent are also found in realistic applications, which might have a negative effect in the aforementioned approaches. As for competitive societies, endevours focus on increase robustness of protocols used in negotiations.

Adaptability is taken into account by proposals in a structural level. Some of them are capable of rearranging functions to adapt to some changes in the tested/monitored MAS. However, these changes are restricted to simple MAS reorganisation (for instance, the death of an agent), while more complex forms of adaptation both in MAS behaviour and structure (such as, coalition formation and emergent behaviours) are not in those techniques' scope. The majority of reviewed work makes no allusion how adaptation or emergent behaviour may influence the performance.

2.6.3 Ease of Use

Designers seek an ideal technique or tool that has only to be plugged into a MAS and starts to properly run. In practice, this effortless tool does not exist yet and thus we analyse the ease of use characteristic from two perspectives: (1) *required information* and (2) *required changes*. On the one hand, every surveyed approach demands for some information about the system so it can certify nominal component functioning. In general, this information can be: *model specification*, which is confronted against gathered information of some aspect of the implemented agent; *interaction pattern* that must be respected by entities (including agents) while communicating (note that they encompass from low-level protocol directives to social commitments); and *performance metric* that concerns dealing with the MAS or the agent in a practical (and frequently domain-specific) manner. On the other hand, some approaches, besides information, also require some level of change in the MAS implementation.

Table 2.3 presents a summary of the required MAS information so the research efforts can properly operate. Across branches, the majority of reviewed endevours demand for a model specification, which can be from an abstract design model of agent functioning and goals to source code containing the library of possible plans. Interaction patterns are also required when fault detection and diagnosis focuses on communication between entities. More specifically, testing advances rely on both the model specification of an agent and interaction protocols, because they use white-box testing that has total information access of the tested component. Moreover, approaches that perform system level testing or validation generally use a performance metric given by the designer to evaluate the overall MAS performance. Counter intuitively, fault tolerance schemes that operate at run-time should require performance metrics to evaluate how well the MAS is working, but at this point they solely maintains the MAS in a nominal service state. In our opinion, these schemes should be also concerned with performance besides proper functioning, which leads towards prognostic methods aiming to forecast trends in the system.

Additionally, some fault tolerance proposals demand modifications in MAS components. For instance, Rustogi et al. [1999], Horling et al. [2000], Mallya and Singh [2005a] require a complete reformulation of the agent internal architecture. Similarly, exception management mechanisms proposed by Platon et al. [2008] need to be inserted as a new component in the agent architecture. Moreover, the remaining proposals requires minimal changes such as: agents must embody some protocols guidelines to communicate with the fault-management architecture, or register to be attached to a sentinel or receive a replica. In general, these proposals that requires minimal changes face more challenges while monitoring, detecting, and treating faults as they must be able to respond to different agent internal functioning.

Table 2.3: The required information from the MAS so the surveyed approach may properly work. Note that only the most relevant publication of the respective project appear below.

Proposed Approach	Model Specification	Interaction Pattern	Performance Metric
Testing in MAS			
– [Knublauch, 2002]	●	○	○
JAT [Coelho et al., 2007]	○	●	○
– [Lam and Barber, 2005a]	●	○	○
– [Pardo et al., 2010]	●	●	●
ACLAnalyser [Serrano et al., 2012]	●	●	○
SEAUnit [Çakirlar et al., 2009]	●	●	●
IDK [Gómez-Sanz et al., 2009]	●	●	○
PDT [Padgham et al., 2013]	●	●	○
eCAT [Nguyen et al., 2012]	●	●	●
Simulation-based Design Validation			
IDE-eli [Sierra et al., 2004]	●	○	●
– [De Wolf et al., 2006]	○	○	●
– [Gardelli et al., 2008]	●	●	○
– [Bernon et al., 2007]	●	●	○
PASSIM [Cossentino et al., 2008]	●	●	●
– [Sudeikat and Renz, 2011]	●	○	○
ELDAMeth [Fortino et al., 2014]	○	●	●
RatKit [Çakirlar et al., 2015]	●	●	●
Fault-Tolerant MAS			
– [Kaminka et al., 2002]	●	○	○
– Kalech [2012]	●	●	○
– [Chia et al., 1998]	●	○	○
– [Horling et al., 2000]	●	○	○
– Micalizio and Torasso [2014]	●	●	○
– [Rustogi et al., 1999]	○	●	○
– [Xu and Deters, 2005]	●	○	○
SaGE [Souchon et al., 2004]	●	○	○
– [Platon et al., 2008]	●	○	○
– [Mallya and Singh, 2005a]	○	●	○
– [Hägg, 1997]	●	○	○
– [Klein et al., 2003]	●	○	○
AAA [Kumar et al., 2000]	○	●	○
DaAgent [Mishra and Xie, 2003]	○	●	○
Chameleon [Kalbarczyk et al., 2005]	●	○	○
Guardian [Miller and Tripathi, 2004]	○	●	○
– [Fedoruk and Deters, 2003]	○	●	○
DARX [Guessoum et al., 2010]	○	●	○

* ● Required; ○ Not Required.

2.6.4 Fault Coverage

As previously discussed in Section 2.1, MASs are susceptible to different types of faults and each of them might be (unintentionally) inserted at any given point of the development life cycle. Research efforts differ with respect to their scope of support (i.e., type of faults they are able to diagnose and handle), and this scope is crucial as designers might want to increase robustness of a particular functionality. We recall Wagner [2000]'s taxonomy in order to disclose fault coverage of the reviewed literature. According to it, faults can be: (1) *program bugs*, errors in programming levels of the systems; (2) *unforeseen states*, omission errors that might be caused by unexpected environmental events; (3) *processor faults*, system or resources crash; (4) *communication faults*, failures of interactions among agents; (5) *emerging unwanted behaviour*, system behaviour that is not foreseen and might happen both on agent or system levels.

The fault coverage of the main surveyed projects is presented in Table 2.4. As this last table reflects aspects previously examined, we are going to perform three distinctive but complementary analyses:

Across branches, one can see that testing in MASs have a more distributed coverage as most of them have three or more types of faults in their scope; simulation-based design validation mainly focuses on pinpointing unwanted behaviours, which is in accordance with they requiring performance metrics; fault tolerance architectures concern with unforeseen states, processor faults, and communication faults because these are the most common faults at run-time.

Across type of faults, one can easily see the predominance of unforeseen states and communication faults. This is intimately related to the required information from MAS as (1) when an unforeseen state happens means that it does not exist in the model specification; and (2) a fault in communication between agents occurs when the interaction pattern is disrespected being either communication protocol or high-level conversation.

Across proposed approaches, one can see that eCAT [Nguyen et al., 2012] has the best fault coverage, followed by Chameleon [Kalbarczyk et al., 2005], DARX [Guessoum et al., 2010], and RatKit [Çakirlar et al., 2015]. As aforementioned, eCAT [Nguyen et al., 2012] tailors techniques to exclusively support BDI architecture designed using the Tropos methodology and, this clear boundary maximises its fault coverage. Both Chameleon [Kalbarczyk et al., 2005] and DARX [Guessoum et al., 2010] are frameworks designed to be technology and application independent and, in the presence of any kind of failure, the former triggers the ARMOR recovery method and the latter simply activates a new agent replica; these mechanisms are transparent to any fault. RatKit [Çakirlar et al., 2015] focuses on collaborative MASs and agents implemented in REPAST; these features constrain the search space for anomalous events while improving fault coverage.

Table 2.4: The fault coverage using Wagner [2000]'s taxonomy. Note that only the most relevant publication of the respective project appear below.

Proposed Approach	Programs Bugs	Unforeseen States	Processor Faults	Communication Faults	Unwanted Behaviour
Testing in MAS					
– [Knublauch, 2002]	√		√		
JAT [Coelho et al., 2007]				√	
– [Lam and Barber, 2005a]	√	√			
– [Pardo et al., 2010]		√	√	√	√
ACLAnalyser [Serrano et al., 2012]			√	√	√
SEAUnit [Çakirlar et al., 2009]		√	√	√	
IDK [Gómez-Sanz et al., 2009]	√	√		√	
PDT [Padgham et al., 2013]	√	√		√	
eCAT [Nguyen et al., 2012]	√	√	√	√	√
Simulation-based Design Validation					
IDE-eli [Sierra et al., 2004]		√			√
– [De Wolf et al., 2006]					√
– [Gardelli et al., 2008]					√
– [Bernon et al., 2007]				√	√
PASSIM [Cossentino et al., 2008]		√		√	√
– [Sudeikat and Renz, 2011]		√			√
ELDAMeth [Fortino et al., 2014]	√			√	√
RatKit [Çakirlar et al., 2015]	√	√		√	√
Fault-Tolerant MAS					
– Kaminka et al. [2002]	√	√			
– Kalech [2012]		√	√	√	
– Chia et al. [1998]					√
– Horling et al. [2000]		√	√	√	
– Micalizio and Torasso [2014]		√	√	√	
– [Rustogi et al., 1999]				√	
– [Xu and Deters, 2005]				√	
SaGE [Souchon et al., 2004]	√	√		√	
– [Platon et al., 2008]		√	√	√	
– [Mallya and Singh, 2005a]		√		√	
– [Hägg, 1997]		√	√		
– [Klein et al., 2003]		√	√		
AAA [Kumar et al., 2000]		√	√		
DaAgent [Mishra and Xie, 2003]			√	√	
Chameleon [Kalbarczyk et al., 2005]	√	√	√	√	
Guardian [Miller and Tripathi, 2004]		√	√	√	
– [Fedoruk and Deters, 2003]	√	√	√		
DARX [Guessoum et al., 2010]	√	√	√	√	

2.7 Summary

Techniques to ensure reliable MASs have become a fundamental component to introduce such systems in real complex (distributed) domains and, therefore, researchers and practitioners are particularly interested in developing these techniques respecting the unique characteristics of MASs. Similarly to other software entities, agents are susceptible to several threats that can jeopardize their correct functioning. The most recent definitions are in rupture to traditional definitions given the autonomy and the social ability of agents. They mainly state that (1) failures in MASs are broader than programming errors, (2) errors spread non-linearly due to interactions, and (3) faults can be exclusively behavioural. After almost two decades of research in MAS, several techniques, architectures, and frameworks have been proposed to protect agent-oriented applications against these threats.

This chapter provides a comprehensive survey and discussion of current research on three of the most important life-cycle phases to improve MAS reliability: testing, simulation-based design validation, and fault tolerant MAS. We presented the MAS testing approaches dividing them by their functioning, namely unit and agent levels, integration, and multi-level testing frameworks. We then discussed efforts that focus on using repeated simulation results to identify misguided design choices. Finally, we described fault tolerance schemes to MAS that were classified in online monitoring and error detection, exception handling systems, sentinel-based architectures, and replication-based architectures. After this broad review of the literature, we were able to proposed an analysis from the designer point of view. This analysis compared aspects that designer would use to choose an existing approach, such as maturity level, MAS feature support, ease of use, and fault coverage. This analysis helped us to identify gaps and envision research trends.

Some of these gaps must be highlighted to better ground our research effort and to explain how we intend to get them bridged. First, as one can see in Section 2.6.3, most approaches require either a model description or an interaction pattern. Contrarily to the literature, the proposed solution requires minimal a priori knowledge given that it builds upon the spectrum-based fault diagnosis. Second, on the one hand, the discussion in Section 2.6.4 concludes that most testing and fault tolerance approaches mainly deal with unforeseen state and communication faults. We focus on unwanted behavioural faults as this level of abstraction better suits run-time recovery of agents. On the other hand, the same discussion shows that simulation-based efforts concentrates on behavioural faults, but are either language or agent architecture specific (Section 2.6.2). As SFL is going to diagnose faults at the agent level, our approach is both language and agent architecture agnostic. We hope that our proposed approach is able to bridge these aforementioned gaps being a relevant contribution to the MAS community.

Chapter 3

Spectrum-Based Fault Localisation for MASs

Previous approaches to guarantee Multi-Agent System (MAS) nominal behaviour (see [Nguyen et al., 2011, Fisher et al., 2007]) assume *a priori* knowledge (i.e., model) to diagnose observed failures (see Chapter 2). This knowledge can be appropriately built when designers fully understand both the environment upon which agents act as well as the agents' state space.

However, in practice, due to (i) the MAS complexity, (ii) environmental dynamism, and (iii) presence of legacy inherent systems, MAS and/or agent models are rather laborious to build. As a consequence, building the model is an error-prone task. Any knowledge not included by designers in the built model may therefore influence the capability of model-based fault diagnosis to effectively recognise faults. The more realistic the problem solved by a MAS becomes, the more complex it gets to minutely encompass all characteristics of related entities in the model. Diagnosis techniques for MASs thus should not completely rely on models built *a priori* to pinpoint unexpected behaviours.

To address this issue, this thesis considers an approach that relies on minimal prior knowledge to pinpoint behavioural faults in MASs. Spectrum-based Fault Localisation (SFL) is a promising technique that does not rely on an explicit model of the system under analysis and has been shown to yield good diagnostic accuracy for software systems [Hofer et al., 2013, Abreu et al., 2009, Le et al., 2013].

The SFL diagnostic process is based on the analysis of the differences in the so-called program spectrum [Reps et al., 1997] (abstraction over program traces) for passed and failed runs. SFL isolates the faulty component, using a *similarity coefficient* as heuristic, whose activity mostly correlates with observed failures. More importantly, SFL can be applied to resource-constrained environments due to its relatively low computational overhead [Abreu et al., 2009, Zoeteweij et al., 2007]. Such properties suggest that SFL is a well-suited technique for MASs. This work aims to study the suitability of SFL as a diagnosis approach in the specific MAS context, proposing the necessary extensions to cope appropriately with the intrinsic characteristics of such a domain.

Traditional uses of SFL have two fundamental limitations when directly applied to MASs. First, the usual abstraction of the program spectrum as the involvement of software components generates a uniform spectrum (with no useful information to discover the fault), since agents are time-persistent entities and are always perceiving and acting upon the environment. The second SFL limitation regards the autonomic facet of the agent. To localise the faulty component, SFL assumes that components output the same results for the same set of inputs; this assumption does not hold for MASs as agents might act differently while facing the same set of events.

The extension described in this work, called Extended Spectrum-based Fault Localisation for MAS (ESFL-MAS), solves the first limitation by proposing the *performance spectrum*, which encodes the agent performance in terms of expected/not expected in a determined time step. This strategy gives a performance-oriented view over the spectrum by tracing agent behaviour expectancy during a run to provide useful diagnostic information and to solve the time-dependency problem. Concerning the second limitation, we have proposed a simple yet elegant solution, which is to execute the monitored MAS several times in order to catch multiple instances of agent behaviours for different environment settings. Additionally, we have suggested an optimisation, named MAS-Filter, which increases diagnostic accuracy by filtering low-entropy rows in the spectrum.

This chapter makes the following contributions:

1. We discuss the limitations of applying SFL with commonly used types of spectrum to time-persistent and autonomous entities such as agents;

2. We describe the Extended Spectrum-based Fault Localisation for Multi-Agent Systems to diagnose agent behavioural faults when testing the system as a whole;

The remainder of this chapter is organised as follows. Section 3.1 discusses main SFL concepts and present an illustrative example for sequential software. Section 3.2 introduces definitions used throughout this chapter. Furthermore, the diagnosis problem for multi-agent systems is defined. Section 3.4 describes the SFL constraints and the proposed changes in order to use SFL for MASs. Finally, Section 3.5 summaries this chapter highlighting its main topics.

3.1 Spectrum-based Fault Localisation

Spectrum-based fault localisation is a dynamic program analysis technique, which requires minimal information about the system to be diagnosed. The binary search algorithm implemented in Java (first column on the left in Table 3.1) will be considered to illustrate the SFL procedure throughout this section. Briefly, this method finds the integer *target* given an ordered integer collection as an array. Binary search divides the sorted collection in half until the sought-for item is found, or until it is determined that the item cannot be present in the smaller collection. For instance, let us assume that this Java source code

Table 3.1: Faulty Java method for binary search. The input for test cases is composed by $collection = \{1, 2, 3, 4, 5\}$ and $target$ as presented below.

Java Source Code	Statement	*target*						
public boolean search (int[ ] *collection*, int *target*) {		null	[ ]	1	2	3	4	5
if (*target* == null) {	1	1	1	1	1	1	1	1
return false; }	2	1	0	0	0	0	0	0
int *low* = 0, *high* = *collection*.length −1;	3	0	1	1	1	1	1	1
while (*low* <= *high*) {	4	0	1	1	1	1	1	1
int *ix* = (*low* + *high*)/2;	5	0	0	1	1	1	1	1
int *rc* = *target*.compareTo(*collection*[*ix*]);	6	0	0	1	1	1	1	1
if (*rc* < 0) {	7	0	0	1	1	1	1	1
high = *ix* − 1;	8	0	0	1	1	0	0	0
} else if (*rc* > 0) {	9	0	0	1	1	1	1	1
//Bug: sign '−' instead of '+'	-	-	-	-	-	-	-	-
low = *ix* − 1;	10	0	0	0	1	0	1	1
} else {	11	0	0	1	0	1	0	0
return true; }	12	0	0	1	0	1	0	0
return false; }	13	0	1	0	0	0	0	0
}	**Error**	**0**	**0**	**0**	**1**	**0**	**1**	**1**

has a bug in the sign of Statement 10[1] where the sign '−' in the equation $low = ix - 1$ should be a '+' sign. Note that this fault can be latent in the system and only leads to errors under certain conditions, but, when it is activated, it leads to an array-out-of-bound exception in the method execution.

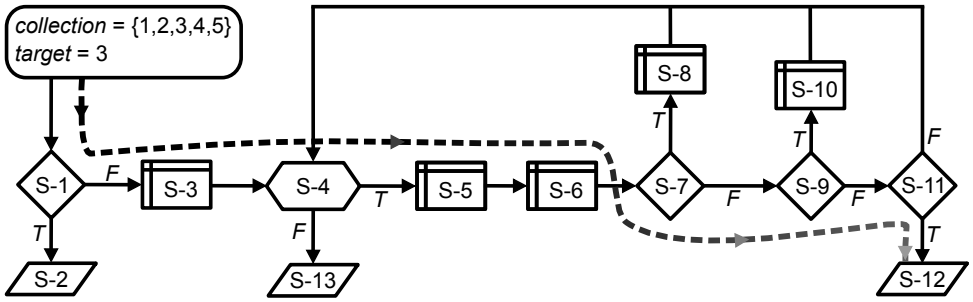
We first need to introduce the concept of *spectrum* to correctly understand SFL. Spectrum is a set of run-time data of the dynamic behaviour exhibited by a piece of software. Literature shows that exist different forms of recording such a spectrum (see Reps et al. [1997], Harrold et al. [1998]). Regardless its form, a spectrum can always be abstracted in terms of two general elements, which are:

- *Component.* It is an element of the system that, for diagnosis purposes, is considered to be atomic. For instance, we consider the statement as the atomic element (i.e., component) of the spectrum in the example.

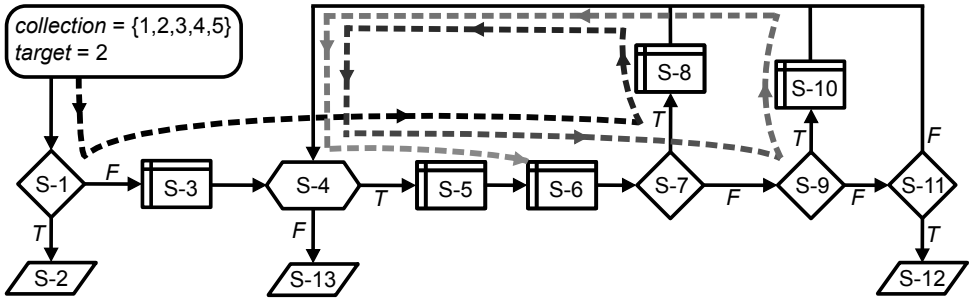
- *Transaction.* It is a specific information about the component. In our illustrative example, this specific information is the involvement of a particular statement. If a particular statement was involved in a given execution, the spectrum is filled with the number 1; otherwise, it is filled with the number 0.

After mapping the software into the aforementioned elements, the next step is to gather

[1]Note that the commented line is not accounted for in SFL since the Java compiler excludes any comment from the binary version.



(a) Execution path for inputs $collection = \{1, 2, 3, 4, 5\}$ and $target = 3$



(b) Execution path for inputs $collection = \{1, 2, 3, 4, 5\}$ and $target = 2$

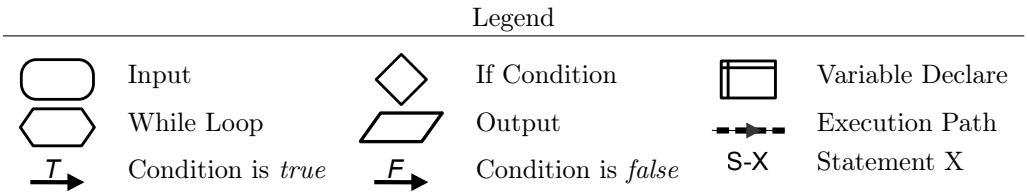


Figure 3.1: Work-flow of the Java code in Table 3.1 with the execution flow of two test cases. It outputs the expected result in (a), whereas it has unexpected output in (b).

run-time profiles containing the specific information about each system component from runs. These runs are execution of a set of test cases that provide different inputs and expected outputs. A test case result can be either nominal ("pass"), representing that the system had the expected output, or an error ("fail"), representing an unexpected software output; this information of in which test case the software failed constitutes another column vector, the *error vector*. Back to our example, let us suppose the method was tested for seven test cases where the input $collection = \{1, 2, 3, 4, 5\}$ is immutable while the $target$ input has values according to presented in Table 3.1. For each of these test cases, the dynamic behaviour of the search method was collected as a statement-hit spectrum and it is shown in the right side of Table 3.1. One must bear in mind that this form of spectrum indicates whether or not a certain code statement was executed given a test case.

Figure 3.1 presents the work-flow of the Java code in Table 3.1 as well as the execution paths for test cases $target = 3$ (Figure 3.1.a) and $target = 2$ (Figure 3.1.b), considering $collection = \{1, 2, 3, 4, 5\}$ for both cases. In the case of a statement-hit spectrum, the process to build the column information for a test case is: as the execution path progresses through the work-flow, positions of involved statements are set to 1, whereas positions of

non-involved statements are set to 0.

As an example, let us build the spectrum's column for both test cases in Figure 3.1. Following the execution path in Figure 3.1.a, the first involved statement is S-1 and consequently the first position of the column is set to 1; then, the next involved statement is S-3 that, at this point of the run, results in [1 0 1] (observe the non-involvement of S-2 represented by 0 in the respective vector position); setting as 1 every involved statement along the execution path, which are: S-1, S-3, S-4, S-5, S-6, S-7, S-9, S-11, and S-12, the final spectrum's column for test case $target = 3$ is [1 0 1 1 1 1 1 0 1 0 1 1 0] as shown in Table 3.1. In the other test case, one can observe the complete different execution path in Figure 3.1.b that involves statements: S-1, S-3, S-4, S-5, S-6, S-7, S-8, S-9, and S-10 (the last two activated after another while cycle); using the same building process, the spectrum's column is [1 0 1 1 1 1 1 1 1 1 0 0 0] as presented in Table 3.1.

Building the error vector requires to compare the expected output and actual software output of the test case; if the output is the expected one, the error vector is set to 0 in the test case position, otherwise, it is set to 1. For Figure 3.1.a, the expected output is *true* because the target element 3 is indeed present in the collection $\{1, 2, 3, 4, 5\}$. Observing the last executed statement S-12 and, given the Java code, one can see that the test case execution returned the expected result; therefore, the error vector is set to 0 in the respective position. Similarly, for Figure 3.1.b, the expected output is also *true* due to the presence of the target element 2 in the collection. However, one can see that the execution path finishes in S-6, which is a variable declaration statement. The fault activation in S-10 results in a negative value of variable ix; ix is then used as index of *collection* in S-6 causing an array-out-of-bounds exception given the attempt to access a negative index in an array. For that reason, the test case has an unexpected output and the error vector is set to 1 in the respective position (see Table 3.1).

After acquiring information for every test case, SFL benefits from both spectrum and error vector to localise the faulty statement. Generically, SFL assumes the hypothesis that closely correlated components are more likely to be relevant to an observed failure. In practice, the basic idea is that comparing transactions over multiple runs and then computing the *suspiciousness* values of components can indicate which of these is the most likely to be the faulty one. Resemblances between binary vector (e.g., error vector) and nominally scaled data (e.g., spectrum) are quantified by means of *similarity coefficients*. These coefficients measure similarity essentially using dichotomy terms, which for our illustrative example refers to the involvement of statement j in the test case i (a_{ij}) and result of test case i (e_i), formally defined as:

$$c_{00}(j) = |\{i|a_{ij} = 0 \wedge e_i = 0\}| \quad (3.1)$$

$$c_{01}(j) = |\{i|a_{ij} = 0 \wedge e_i = 1\}| \quad (3.2)$$

$$c_{10}(j) = |\{i|a_{ij} = 1 \wedge e_i = 0\}| \quad (3.3)$$

$$c_{11}(j) = |\{i|a_{ij} = 1 \wedge e_i = 1\}| \quad (3.4)$$

where, the $c_{11}(j)$ is the number of failed runs in which statement j is involved, $c_{10}(j)$ is the number of passed runs in which statement j is involved, $c_{01}(j)$ is the number of failed

Table 3.2: The values of dichotomy terms for SFL example.

Dichotomy Element	Statement												
	1	2	3	4	5	6	7	8	9	10	11	12	13
c_{11}	3	0	3	3	3	3	3	1	3	3	0	0	0
c_{10}	4	1	3	3	2	2	2	1	2	0	2	2	1
c_{01}	0	3	0	0	0	0	0	2	0	0	3	3	3
c_{00}	0	3	1	1	2	2	2	3	2	4	2	2	3

runs in which statement j is not involved, and $c_{00}(j)$ is the number of passed runs in which statement j is not involved.

Aiming at illustrating the mechanism of filling dichotomy terms, let us focus on the column of Statement 8, which is [0 0 1 1 0 0 0], and correlate it with the error vector [0 0 0 0 0 1 1]. First, second, and fifth positions of both vectors have value 0, therefore $c_{00} = 3$. In the third position, involvement column has value 1 whereas error vector has value 0, producing $c_{10} = 1$. Forth position of both vector has value 1, thus $c_{11} = 1$. Finally, $c_{01} = 2$ because, in sixth and seventh positions, involvement column has value 0 and error vector has value 1. Table 3.2 shows all dichotomy terms for every statement of our example.

Let us finish the diagnosis process of our example by using the Jaccard coefficient (C_{16}) to compute the suspiciousness value of each statement (see Table 3.3). As an example, we substitute dichotomy values of Statement 8 into Jaccard formulae as illustrated below:

$$C_{16} = \frac{c_{11}}{c_{11} + c_{10} + c_{01}} = \frac{1}{1 + 1 + 2} = 0.25 \tag{3.5}$$

For the illustrative example, by computing suspiciousness values and ranking statements with respect to them (see Table 3.3), SFL (correctly) identifies Statement 10 as the most likely location of the fault. Clearly, this example has a small number of test cases and statements; nonetheless, it fully illustrates the SFL diagnostic process.

Table 3.3: The Jaccard similarity coefficient values and ranking for SFL example.

Statement	1	2	3	4	5	6	7	8	9	10	11	12	13
Coefficient Value	0.43	0.00	0.50	0.50	0.60	0.60	0.60	0.25	0.60	1.00	0.00	0.00	0.00
Ranking (D)	8	10	6	7	2	3	4	9	5	1	11	12	13

In a nutshell, SFL is a diagnosis technique that relies on dynamic analysis of the system, not requiring any additional modelling effort. It assumes that exists a high correlation between the fault activation and the system failure. System components are ranked with respect to their suspiciousness values computed using predefined heuristics. SFL requires short time to compute a diagnostic report as it:

1. Initialises the dichotomy terms: $\mathcal{O}(1)$.

2. Computes the suspiciousness value per component for N test cases: $\mathcal{O}(M \times N)$.

3. Ranks components of M according to their suspiciousness values: $\mathcal{O}(M \times log\,(M))$.

Therefore, the overall time complexity is $\mathcal{O}(M \times N + M \times log\,(M))$. The space complexity is $\mathcal{O}(M \times N)$ for storing the spectrum and $\mathcal{O}(M)$ for storing the diagnostic report. Since $\mathcal{O}(M \times N)$ has a faster growth than $\mathcal{O}(M)$, the latter can be disregarded.

3.2 Concepts and Definitions for SFL in MAS

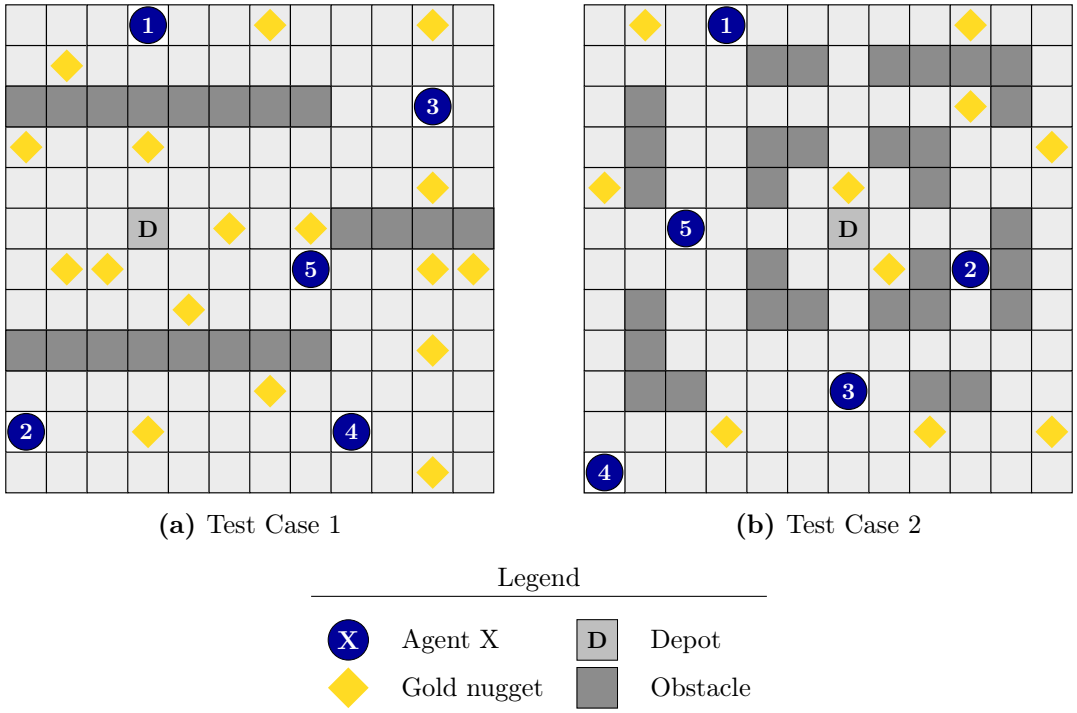
A multi-agent system is a computational system composed by a set of agents. Each of them (agents) is able to autonomously reason as well as sense and act upon a certain environment, leading to the satisfaction of its own (and sometimes shared) goals. Agents are time-persistent entities, that is, they are continuously operating to fulfil their goals.

Definition 1 (Multi-Agent System)**.** *A multi-agent system MAS consists of a set of agents* $AGS = \{Ag_1, \cdots, Ag_m, \cdots, Ag_M\}$ *that are situated in an environment* E*. Thus, a MAS is a tuple* $MAS = \langle E, AGS \rangle$ *[Lettmann et al., 2011].*

The MAS runs during a limited time interval, which can be discrete (represented by non-negative integers) or continuous (represented by real numbers) [Bhattacharya and Waymire, 1990]. This thesis considers the *discrete-time* representation because (1) continuous time can be sampled and (2) we intend to work within the boundaries of finite-dimensional events[2]. Throughout this thesis, we refer to an unit of time either using *time frame* or *time step*.

Example 1 *In order to explain the basic concepts and the application of spectrum-based fault localisation in MASs, we make use of a running example. Figure 3.2 shows this example MAS which is borrowed from our experimental setup that will be thoroughly described in* **??***. For the sake of clearness, we have reduced the number of agents and size of the environment of this example. In a nutshell, agents find themselves exploring an area searching for the gold nuggets spread over the environment; they aim to collect as much gold nuggets as they can and to deliver them to a depot where the nuggets are safely stored. Let us assume that agent* Ag_5 *erroneously compute its distance from a gold nugget due to an unforeseen bug in the reasoning process unintentionally left by the designer/programmer; as a result of this bug,* Ag_5 *has lower performance in some specific situations than it should have. Throughout this chapter, it is shown how to use SFL to pinpoint the faulty agent.*

[2] A finite-dimensional event depends only on the values of the process at finitely many time points [Bhattacharya and Waymire, 1990].



(a) Test Case 1 **(b)** Test Case 2

Figure 3.2: Goldminers example used as the illustrative example throughout this chapter.

After defining the basic MAS elements, it is needed to assess its performance by measuring observable variables of both the environment and the set of agents. For instance, a MAS aiming to control traffic lights uses environment variables such as traffic flow to measure MAS performance. Likewise, utility-based MASs measure the global utility (which is commonly used as a system performance measure) by summing the utility of all agents, that is, applying a function over observable variables of agents. In our running example, the MAS performance can be measured by the amount of gold nuggets in the depot.

Definition 2 (Measurable Space)**.** *A* measurable space *is a set Z, together with a non-empty collection $\mathcal{Z}$, of subsets of Z, satisfying the following two conditions:*

1. *For any X, Y in the collection $\mathcal{Z}$, the set $X \cap Y^c$* [3] *is also in $\mathcal{Z}$.*

2. *For any $X_1, X_2, \cdots \in \mathcal{Z}$.*

The elements of $\mathcal{Z}$ are called *measurable* sets of variables.

Definition 3 (Measure of *MAS*)**.** *Let $(E, \mathcal{E})$ and $(AGS, \mathcal{AGS})$ be two measurable spaces. A* measure *of MAS performance consists of a two non-empty subsets $M_E \subset \mathcal{E}$ and $M_{AGS} \subset \mathcal{AGS}$ together with $\mu : f\left(\langle \mathcal{E}, \mathcal{AGS} \rangle, n\right) \to \mathbb{R}$ that maps the performance of a MAS in time n (considering discrete time) to a real number. It uses as arguments the measurable set of variables of the tuple $\langle E, AGS \rangle$.*

[3] $X \cap Y^c$ is the set of all values of X that are not in Y

Note the strong connection between MAS and environment (Definition 1) termed by Ferber [1999] as "the agent/environment duality". Thereby, we formally define our understanding of an environment, which introduces the necessary components to define test cases for MASs.

Definition 4 (Environment)**.** *An environment E is described as a tuple $E = \langle S, s_0, \mathcal{A} \rangle$ where*

- *$S = \{s_0, s_1, \cdots\}$ is a countable set of environment states with an initial state s_0.*
- *$\mathcal{A} = \times_{Ag_m \in AGS} \mathcal{A}_{Ag}$ denotes all joint actions (of agents) that can be performed in the environment.*

Definition 4 does not explicitly introduce sets of environment objects other than agents, since all (observable) information of these objects are considered to be incorporated into the set of environment states S. In our example, the depot and positions of gold nuggets are objects that are modelled within the environment state set.

A test case, in ordinary sequential programs, comprises input values and excepted output values. Defining the input and output of MASs is straightforward by means of the environment and the measure of MAS.

Definition 5 (Input, Output)**.** *Given a multi-agent system MAS situated in an environment E and measured by μ, then the* inputs *comprise the initial state s_0 of E and initialisation parameters for agents (ags_0). The* outputs *comprises the measure μ of MAS.*

With this definition of input and output we are able to define a test case for a MAS and its evaluation.

Definition 6 (Test case)**.** *Given a multi-agent system MAS, then a tuple $\langle I, O \rangle$ is a* test case *for MAS if and only if:*

- *I is a set of values for each object specifying the environment state s_0 and a set of initialisation parameters ags_0 for every agent $Ag_m \in AGS$.*
- *O is a set of values specifying expected MAS outputs when measured by μ.*

In our setting, test case evaluation works as follows. First, the environment is initialised with state s_0 and agents with parameters ags_0. Subsequently, the MAS is executed. The outputs computed by μ are compared with the expected values stated in the test case. If the computed output value is not equivalent to the expected value, the MAS fails the test case during that specific period. Otherwise, the MAS passes the test case during a specific period. Note that, unlike in "traditional" software, running a MAS with a test case yields a vector of errors where each element is the passed/failed MAS status at time n.

Example 2 *A test case for our running example from Figure 3.2.a is* $I = \{$ *depot* $= (4,6)$, *gold* $= \{ (1,4), (2,2), (2,7), (3,7), (4,4), (4,11), (5,8), (6,6), (7,1), (7,10), (8,6), (11,1), (11,5), (11,7), (11,9), (11,12), (12,7)\}$, *obstacle* $= \{(1,3), (2,3), (3,3), (4,3), (5,3), (6,3), (7,3), (8,3), (1,9), (2,9), (3,9), (4,9), (5,9), (6,9), (7,9), (8,9), (9,6), (10,6), (11,6), (12,6)\}$, $ags_0 = \{(4,1), (1,11), (11,3), (9,11), (8,7)\}\}$ *and* $O = (1 * collectedgold)/n$. *The* MAS *has to be executed during a period so it is possible to establish when it passes and/or fails the test case.*

Avižienis et al. [2004] define fault-related terms as: a *failure* is an event that occurs when delivered service deviates from correct service; an *error* is a system state that may cause a failure; and a *fault* is the cause of an error in the system. In this thesis, we consider that faults in agents' behaviour depend on a given context, i.e. on how each agent interprets that particular situation [Platon et al., 2007]; and, mainly, these faults are a systemic matter as it might affect the overall performance [Klein et al., 2003]. More specifically, the term "faulty agent" is used to refer to an agent that either is not healthy and needs to be repaired or has been induced to a failure state (known as cascading effect).

Diagnosis is the task of pinpointing the faulty component that led to symptoms (failure/error). In software, the set of components can be defined at any granularity level: a class, a function, or a block. The lower the granularity level gets, the more focused is the diagnosis; even though such low granularities require more computational effort [Zamir et al., 2014]. Hence, following Reiter's formalism [Reiter, 1987], the diagnosis problem for MASs can be defined as follows.

Definition 7 (Multi-Agent Diagnosis Problem)**.** *Given a multi-agent system MAS that has a set of agents AGS and a set of observations OBS for test case* $\langle I, O \rangle$*, then the diagnosis problem is to find the faulty agent which is responsible for the mismatch between the expected MAS performance and the observed one.*

The multi-agent diagnosis problem is defined with granularity at the agent level and thus considering agents as black boxes. This is a fair assumption when different parties implement agents reasoning and do not completely share their knowledge and/or architecture. On the one hand, the technique proposed in this chapter is not able to identify the specific bug inside the code of the faulty agent; on the other hand, however, it has the advantage of being agnostic to programming languages and agent architectures.

3.3 Limitations of SFL

As stated by Definition 7, this work deals with agent-level diagnosis, because it focuses on testing the MAS as a whole ensuring that agents' performance is nominal when facing several environmental conditions. We envision that, at run-time, the agent-level is the most advisable one as faulty agents are simply removed from the running MAS to avoid performance degradation. Therefore, component abstraction discussed in Section 3.1 is mapped to *agents*. The challenge when applying SFL in MASs is to map the transaction abstraction so the diagnostic process have useful information about agent to be able to

pinpoint the fault. Current SFL approaches have limitations and such limitations will be discussed in this section with the purpose to ground the proposed extensions.

3.3.1 Time Persistence

A limitation of the discussed SFL system assessment approach is related to the assumption that every test case is only function of the input variables [Abreu et al., 2009]. While a test case indeed solely depends on inputs when diagnosing non-time-persistent software, such an abstraction is unable to represent MAS performance as it is measured during some period (see Definition 3). For instance, let us assume that for both test cases in Figure 3.3 the expected output is have an empty depot at $n = 3$. As it can be seen, this expectation is met, thus the error vector is $[0, 0]$. One can see that unexpected MAS outcomes before $n = 3$ are neglected as they are not encoded in the error vector. Disregarding the effect of time on the MAS performance implies that the perceived system degradation may be completely overlooked by the diagnostic algorithm and, consequently, the diagnostic quality is negatively affected.

Moreover, the hit spectrum is the far most commonly used type of spectrum [Harrold et al., 1998]. It encodes the component's activity in terms of involved/not involved in a given test case. A limitation of the SFL approach presented in Section 3.1 is related to high level of abstraction enforced by hit spectrum as it does not provide useful information about the state of the agent during an execution.

Since agents are time-persistent entities, they are always active and acting upon the environment; this creates spectra with very low entropy. Low entropy in the spectrum means that there is less useful information in the spectra and, consequently, decreasing SFL diagnostic quality [Gonzalez-Sanchez et al., 2011, Campos et al., 2013]. Therefore, block hit spectra is not suitable to MASs. This conclusion can be generalised to other types of spectrum that do not account for time within transactions.

Using a simpler version of our running example presented in Figure 3.3 to illustrate this limitation, consider that agents' activities are encoded as hit spectra. Every agent at $n = 1$ has acted upon the environment, thus its row in the hit spectrum is $[1, 1, 1]$. Following, Ag_1, Ag_2, and Ag_3 perform a *Move Left*, *Pick*, and *Move Up* actions respectively at $n = 2$, resulting also in a $[1, 1, 1]$ row. This reasoning follows throughout both MAS runs, creating hit spectra fulfilled with 1's, thus impairing the diagnostic accuracy.

Time persistence is not specific to MASs; on the contrary, it is intrinsic to well-known software systems such as web servers. In contrast to our performance spectrum that effectivily encode agents' activity during some time, Casanova et al. [2013] deal with diagnosing architecture run-time failures aiming at self-healing systems. Authors point out that (similarly to our case) traditional proposals of transaction do not correctly encodes the system's time progression. Their solution is to map the transaction as a two lower-level events: a message sent from the origin and a message arrived at destination. However, this solution is not sufficient to MASs because of two reasons. First, agents do not necessarily have to interact with each other to achieve their goal(s); they choose whether to do so. In the case of Casanova et al. [2013], interactions among components are fundamental to the whole

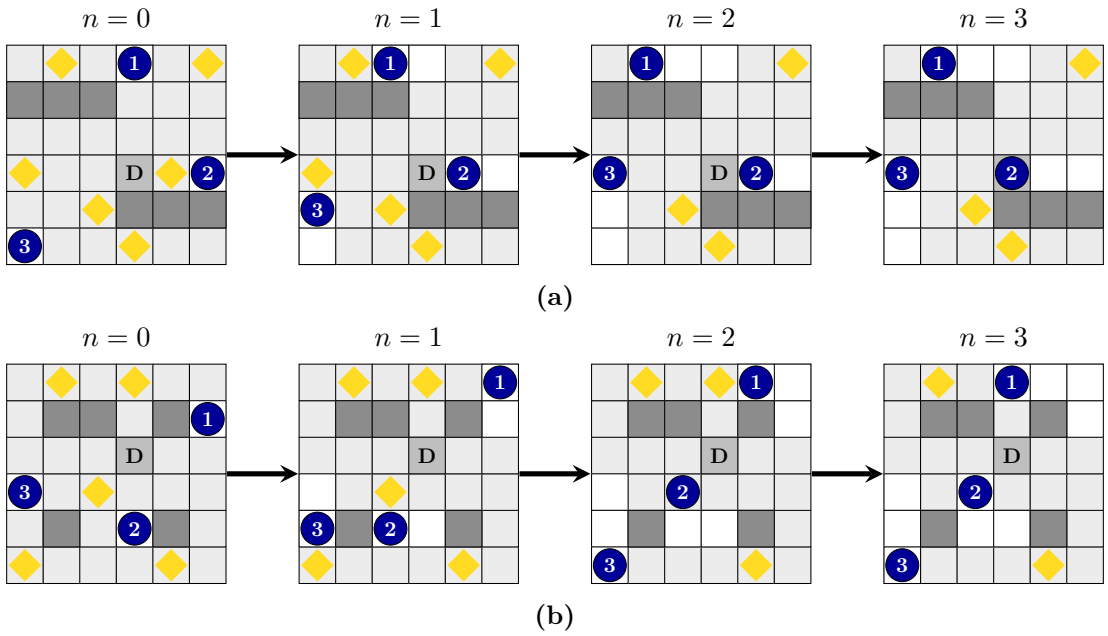


Figure 3.3: A small version of Goldminers to demonstrate the time-related limitations.

architecture. Therefore, the aforementioned extensions are required to apply SFL in MAS even though there exists SFL applications in systems (up to a point) similar to MASs.

3.3.2 Agent's Autonomy

Another limitation of the SFL approach is related to the assumption that every test case produces the same activity matrix and error vector. This is fair assumption when dealing with reactive software such as the Java method for binary search (Table 3.1). The method is going to have the same outcome for the same input arguments. Such assumption does not hold for MAS because agents autonomously decide their next action based on their perceptions; as so, even facing the same perceptions and events, agents may choose an action different from their previous one. As a consequence, the discussed SFL approach is unable to catch multiple instances of the agent's behaviour.

The presence of autonomous behaviour in the system implies that an single environment setting (i.e., test case) derives multiple possible time lines. In such time lines, it is likely that neither the system or agents have the exactly same performance, but rather stay within an expected one (assuming nominal functioning). Consider the test case presented in Figure 3.4 as an illustrative example of this time-line derivation. Even having the same initial configuration at $n = 0$, scenario at $n' = 1$ and $n'' = 1$ are completely different as agents choose to pursue different gold nuggets. For instance, Ag_1 proactively moves towards the gold nugget in the left at $n' = 1$ and in the right at $n'' = 1$, simply relying on its own perceptions and decision making.

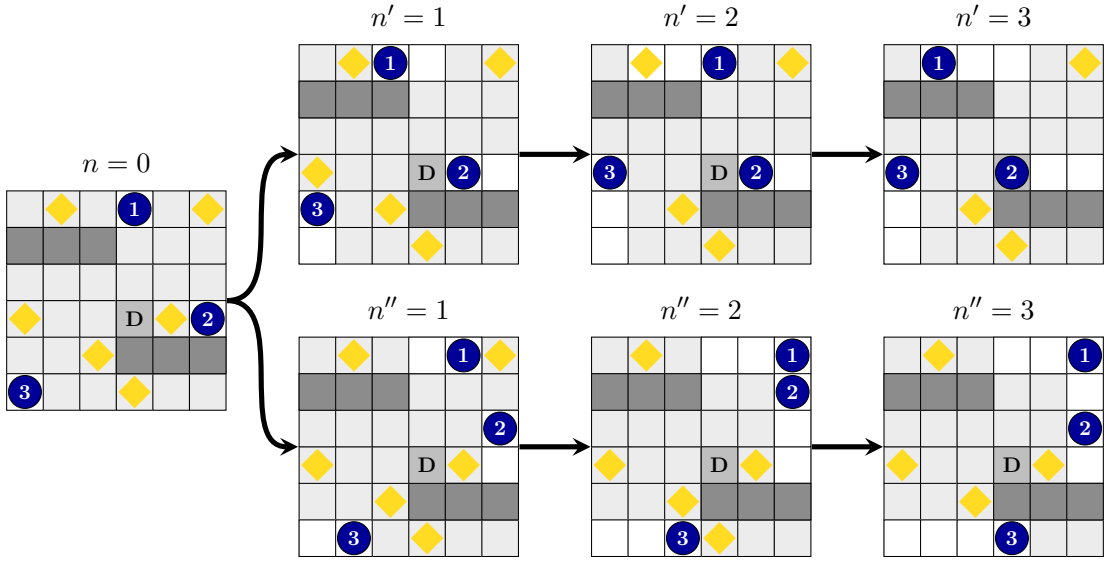


Figure 3.4: A small version of Goldminers to demonstrate the autonomy-related limitation. One can see that a single initial configurations (i.e., test case) may derive multiple time lines, two time lines in this specific example.

3.4 Extending SFL for Multi-Agent Systems

In this section, we describe our approach called Extended Spectrum-based Fault Localisation for MAS. We discuss the proposed solution for each of the aforementioned limitations so spectrum-based diagnosis can be applied to MASs.

Since MASs must run during some period (several time frames) to observe their emerging behaviour (first limitation discussed on Section 3.3.1), As a solution, we propose that both spectrum and error vector are built with time-frame granularity (requiring a different type of instrumentation); hence, providing a fine grained performance information that enhances ESFL-MAS diagnostic quality.

The solution to overcome the second limitation on Section 3.3.1 is to encode each agent's performance in terms of expected/not expected during a time step. Each agent $Ag_m \in AGS$ is associated with a health variable h_m. An agent's health state is either expected (when it is performing as expected) or unexpected (when it is not performing as expected). A *detection of symptoms* (also called *error detection*) phase is responsible to infer any behavioural violation from observations of the system [de Kleer and Williams, 1987]. Specifically for agents, this can be done using several methods from monitoring agent's utility to applying anomaly detection techniques [Chandola et al., 2009, Khalastchi et al., 2015]. Note that detecting an unexpected behaviour does not necessarily mean that one has identified the agent that is causing the system failure to occur [Kalech and Kaminka, 2011]. ESFL-MAS pinpoints the faulty agent so the designer is able to fix it, which is also essential to improve reliability of MASs. Given this performance-oriented perspective, we propose the *performance spectrum*, where the error detection phase generates the set of data composing the spectrum. It is worthwhile mentioning that error detection mechanisms are outside the scope of this thesis.

Definition 8 (Performance Spectrum)**.** *Let N denote the number of passing and failing time frames. Let N_f and N_p, $N_f + N_p = N$, denote the number of fail and pass sets (spectra), respectively. The* performance spectrum *abstraction encodes the agents' activity in terms of expected/unexpected behaviour at the nth time frame and the* error vector *(i.e., transactions' correctness) in terms of MAS performance has passed/failed the test case at nth time frame. Using the performance spectrum, A and e are defined as:*

$$A_{nm} = \begin{cases} 0, & \textit{if agent } m \textit{ has acted as expected at time frame } n \\ 1, & \textit{otherwise} \end{cases} \tag{3.6}$$

$$e_n = \begin{cases} 0, & \textit{if MAS performance has passed at time frame } n \\ 1, & \textit{otherwise} \end{cases} \tag{3.7}$$

Given that the pair (A, e) highly depends on both the *environment settings* and the *agents' autonomy*, SFL is limited to catch multiple instances of both dependencies. This limitation is addressed as follows. First, to solve the problem of multiple environment settings, one must run the MAS for different environment and agent settings; thus, the ESFL-MAS collects performance spectra referring to several test cases. Second, to solve the agent's autonomy problem, one must execute the MAS J rounds of the same test case to ensure that the collected spectra cover as many agents' activation paths (i.e., choices) as possible.

Definition 9 (Collection of Performance Spectra)**.** *A* collection of performance spectra *for a test case i is denoted by $PS_i = \{(A, e)_{i,1}, \cdots, (A, e)_{i,j}, \cdots, (A, e)_{i,J}\}$, which has been executed J rounds. Additionally, $PS = \{PS_1, \cdots, PS_i, \cdots, PS_I\}$ denotes the collection of all performance spectra, where I is the number of available test cases.*

Example 3 *We execute two rounds of the* MAS *of Example 1 for Test Cases 1 and 2 (see Figures 3.2.a and 3.2.b), i.e., $I = 2$ and $J = 2$. Assuming there is an error detection mechanism able to detect unexpected behaviour in the agents, the collection of performance spectra $PS = \{(A, e)_{1,1}, (A, e)_{1,2}, (A, e)_{2,1}, (A, e)_{2,2}\}$ presented in Figure 3.5 (ignoring the highlighted columns that will be explained later on) is built.*

We have observed that agents are constantly monitored over time but do not consistently fail. For this reason, the collected performance spectra have several time frames in which either every agent performed an expected behaviour or every agent performed an unexpected behaviour. Both events contain no information for SFL because only the variability of transactions in the spectrum contributes towards improving diagnostic quality. A proposed optimisation to ESFL-MAS, named MAS-Filter, recognises these aforementioned events and filters them from the performance spectra to increase quality of the diagnosis process. Conceptually, when excluding the non-useful time frames, the entropy value of the spectra tends to its optimal value and consequently increases diagnostic accuracy. The MAS-Filter is defined as follows:

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	0	1	1	1	1
2	1	1	1	1	1	0
3	1	1	0	0	0	1
4	0	0	0	0	0	0
5	0	0	0	0	1	1

(a) $(A, e)_{1,1}$

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	1	1	1	1	1
2	1	1	1	1	0	0
3	0	1	0	1	0	0
4	0	0	0	0	0	1
5	0	1	0	0	1	1

(b) $(A, e)_{1,2}$

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	0	0	0	0	0	0
2	1	0	1	1	0	0
3	0	1	0	1	1	1
4	0	0	1	0	1	1
5	1	1	1	1	1	1

(c) $(A, e)_{2,1}$

n	A					e
	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5	
1	1	0	1	1	0	0
2	1	0	0	1	1	1
3	1	1	0	1	1	1
4	0	0	0	0	1	0
5	0	0	0	0	0	0

(d) $(A, e)_{2,2}$

Figure 3.5: Collection of Performance Spectra for both Test Cases 1 and 2 ($I = 2$) with $J = 2$

$$\text{MAS-Filter}(A, e) = (A_x, e_x) : x = \{n \mid \exists i, j : A_{ni} \neq A_{nj}\} \tag{3.8}$$

The filter is based on the set x, which is the set of all spectrum's rows for which exist elements with both values 1 and 0. Intuitively, it filters time steps in which all agents have expected (or unexpected) behaviours detected.

Example 4 *Following our running example, the MAS-Filter excludes the highlighted rows of the four pairs (A, e) shown in Figure 3.5, creating a filtered version for each of them that has higher entropy than the original spectrum.*

For each agent, a dichotomy matrix is then created (see Table 3.4). One dimension of this matrix is related to the amount of time steps in which the agent had an unexpected behaviour detected, and the other is the passed/failed MAS status determined by the test case expected output. Concisely, c_{11} is the number of time steps in which an agent performed an unexpected behaviour and MAS was detected as failed, c_{10} is the number of time steps in which an agent performed an unexpected behaviour and the MAS was considered correct, c_{01} is the number of time steps in which an agent did not perform an unexpected behaviour and the MAS was found faulty, and c_{00} is the number of time steps in which an agent did not perform an unexpected behaviour and the MAS was detected correct. Formally,

Table 3.4: Dichotomy table for performance spectrum

	Behaviour of Ag_m	
MAS status	Unexpected ($A_{nm} = 1$)	Excepted ($A_{nm} = 0$)
Failed ($e_n = 1$)	c_{11}	c_{01}
Passed ($e_n = 0$)	c_{10}	c_{00}

$$c_{00}(m) = |\{n|A_{nm} = 0 \wedge e_n = 0\}| \quad (3.9)$$
$$c_{01}(m) = |\{n|A_{nm} = 0 \wedge e_n = 1\}| \quad (3.10)$$
$$c_{10}(m) = |\{n|A_{nm} = 1 \wedge e_n = 0\}| \quad (3.11)$$
$$c_{11}(m) = |\{n|A_{nm} = 1 \wedge e_n = 1\}| \quad (3.12)$$

Each element in the table is a counter that will be used to calculate suspiciousness values in ESFL-MAS.

Example 5 *Figure 3.6 presents the dichotomy tables of our running example. Each dichotomy term is computed only taking into account the non-highlighted rows of the performance spectra in Figure 3.5.*

SFL abstractly models each agent using a *weak-fault model*, meaning that the spectrum-based diagnosis essentially consists in identifying the agent whose transactions resembles the error vector the most. The suspiciousness values calculated by similarity coefficient quantify these resemblances, under the assumption that a high similarity with the error

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	2	1	1	1	2
c_{10}	0	0	0	0	0
c_{01}	1	2	2	2	1
c_{00}	0	0	0	0	0

(a) $(A, e)_{1,1}$

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	0	1	0	0	1
c_{10}	1	2	1	2	0
c_{01}	1	0	1	1	0
c_{00}	1	0	1	0	2

(b) $(A, e)_{1,2}$

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	0	1	1	1	2
c_{10}	1	0	1	1	0
c_{01}	2	1	1	1	0
c_{00}	0	1	0	0	1

(c) $(A, e)_{2,1}$

	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
c_{11}	2	1	0	2	2
c_{10}	1	0	1	1	0
c_{01}	0	2	2	1	1
c_{00}	1	0	1	0	1

(d) $(A, e)_{2,2}$

Figure 3.6: Dichotomy tables for the running example

vector indicates a high suspiciousness for that agent to have caused the detected failure. SFL process can be broadly divided as diagnostic candidate generation and ranking [Abreu and Van Gemund, 2010] resulting in the diagnostic report.

Definition 10 (Diagnostic Report)**.** *A* diagnostic report *is an ordered set D of agents sorted in respect with their suspiciousness of being the true fault explanation.*

Example 6 *Let us continue to identify the faulty agent of our running example. ESFL-MAS uses the information of dichotomy tables from Figure 3.6 to compute suspiciousness values using the similarity coefficient. In this example, we choose the Jaccard coefficient (C_{16}). ESFL-MAS computes the suspiciousness value by inserting the information of dichotomy terms into the formula, e.g., for agent Ag_5:*

$$C_{16} = \frac{c_{11}}{c_{11} + c_{10} + c_{01}} = \frac{7}{7 + 1 + 1} = 0.78 \tag{3.13}$$

The process is repeated for every agent until it obtains the values shown in Table 3.5. Afterwards, the list of agents is sorted in descending order with respect of suspiciousness values. As expected, the faulty agent Ag_5 was the highest ranked by ESFL-MAS in the end of the process.

Table 3.5: The Jaccard similarity coefficient values and diagnostic report for the running example.

Agent	Ag_1	Ag_2	Ag_3	Ag_4	Ag_5
Coefficient Value	0.36	0.36	0.18	0.31	0.78
Rank	2	2	4	3	1

Algorithm 1 mathematically describes the ESFL-MAS procedure. The goal of the algorithm is to identify the most probable faulty agent in given MAS based on their detected behavioural violations. It takes as input a collection of performance spectra PS and returns a diagnostic report. In contrast to the SFL algorithm, the set of performance spectra is built using an error detection mechanism that is independent of ESFL-MAS and it is outside the scope of this thesis as well.

In practice, the algorithm works by iteratively selecting an spectrum (A, e) from PS (line 2) and then filtering the non-useful rows through MAS-Filter (line 3). This filtered performance spectrum (A', e') is used to feed the dichotomy terms (lines 4 and 5), which are the input to the similarity coefficient s (line 6). ESFL-MAS returns the diagnostic report with the most probable faulty agent based on the suspiciousness values of each agent (lines 7 and 8).

The algorithm of MAS-Filter (line 3 in Algorithm 1) is presented in Algorithm 2. It iterates over the number of time steps N and the number of agents M (lines 3 to 5) and check whether A_n has a difference among its elements (line 6). If it has, the control flag *skip* is set to false (line 7) and the search for a different element is interrupted. Finally, the pair (A_n, e_n) is appended to filtered spectrum if *skip* is set false (lines 11 and 12).

Algorithm 1 ESFL-MAS

Inputs: PS

Output: Diagnostic report D

1 $c = [0]_{2\times 2\times M}$
2 **for** $(A_{N\times M}, e_{N\times 1}) \in PS$ **do**
3 $\quad (A', e') \longleftarrow (A_x, e_x) : x = \{n \mid \exists i, j : A_{ni} \neq A_{nj}\}$
4 $\quad c'_{pqm} \longleftarrow |\{n \mid A'_{nm} = p \wedge e'_n = q\}| : (p \in \{0,1\}, q \in \{0,1\}, m \in \{0, \cdots, M-1\})$
5 $\quad c \leftarrow c + c'$
6 $d \longleftarrow s(c)$
7 $D \longleftarrow \text{Sort}(d)$
8 **return** D

The algorithm for computing the dichotomy terms of each agent (line 4 in Algorithm 1) is presented in Algorithm 3. It works by iterating over all elements in A (lines 2 and 3), thus adding 1 to the previous value in the dichotomy table c (lines 4 to 6). This algorithm is the heuristic-based SFL's core. Both Algorithms 2 and 3 were presented separately for the sake of clearness. In practice, the MAS-Filter application and the dichotomy terms feeding were done inside the same cycle that iterates over N and M.

The upper-bound values for the complexity of ESFL-MAS algorithm is $\mathcal{O}(I \times J \times M \times N)$. The space complexity is $\mathcal{O}(I \times J \times M \times N)$ for storing the collection of performance spectra. Even though ESFL-MAS is more complex than original SFL, it is still solvable in polynomial time and can be considered light weighted.

Algorithm 2 MAS-Filter

Inputs: (A, e)

Output: Filtered performance spectrum (A', e')

1 $A' = [0]_{N\times M}$
2 $e' = [0]_N$
3 **for** $n \in \{0, \cdots, N-1\}$ **do**
4 $\quad$ *skip* $\longleftarrow$ **true**
5 $\quad$ **for** $m \in \{0, \cdots, M-2\}$ **do**
6 $\quad\quad$ **if** $A_{nm} \neq A_{n(m+1)}$ **then**
7 $\quad\quad\quad$ *skip* $\longleftarrow$ **false**
8 $\quad\quad\quad$ **break**
9 $\quad$ **if** *skip* **then**
10 $\quad\quad$ **continue**
11 $\quad A'_n \longleftarrow A_n$
12 $\quad e'_n \longleftarrow e_n$
13 **return** (A', e')

Algorithm 3 feedDichotomyTerms

Inputs: (A, e)

Output: Dichotomy table c

```
1 c = [0]_{2×2×M}
2 for n ∈ {0, ··· , N − 1} do
3     for m ∈ {0, ··· , M − 1} do
4         p = A_{nm}
5         q = e_n
6         c_{pqm} = c_{pqm} + 1
7 return c
```

3.5 Summary

MASs are constantly susceptible to several threats that can jeopardise their nominal performance. To detect any abnormality on agent behaviours is a paramount step to ensure performance; however, in practice, error detection rarely exhibits 100% precision. Moreover, when there are multiple cooperative agents, errors might be masked by other agents and possibly go undetected. As stated by [Kalech and Kaminka, 2011], "diagnosis is an essential step beyond the detection of the failures. Mere detection of a failure does not necessarily lead to its resolution." Motivated by this, we devised and discussed an approach, called ESFL-MAS, which is able to identify agents that may jeopardise the overall performance through run-time profiles of the system while requiring minimal information about it.

We mapped MAS concepts to basic elements of SFL, being agents and their expected behaviour at given time respectively mapped to components and transactions, thus incorporating time within the spectrum. Diagnosis at the system level steer the proposed technique towards a performance-oriented direction, resulting in the so-called performance spectrum. Lastly, some data do not effectively contributed to localise the faulty agent given the intensive monitoring of agents; thus we suggested MAS-Filter that increases diagnostic accuracy by excluding low-entropy rows in spectra.

Chapter 4

Empirical Evaluation of Similarity Coefficients for ESFL-MAS

Literature has shown that there is no standard similarity coefficient that yields the best result for SFL [Yoo et al., 2014, Hofer et al., 2015, Le et al., 2013]. Empirical evaluation is therefore essential to establish which set of heuristics excels for the specific context to which SFL is being applied. To the best of our knowledge, SFL has not as yet been applied to diagnose behavioural faults in MASs; hence, there is the need to empirically evaluate different formulae using known faults to compare the performance yielded by several similarity coefficients. This chapter empirically determines the best similarity coefficients for SFL including the necessary extensions in the MAS context.

Our experiments study an exhaustive list of similarity coefficients and, as some of them behave very similar, a clustering analysis was performed. Clusters were formed based on the diagnostic performance of heuristics. The performance of similarity coefficients was also assessed regarding the influence of quantity of available data to the Extended Spectrum-based Fault Localisation for Multi-Agent Systems (ESFL-MAS) and, due to ESFL-MAS dependence on the error detection phase, the impact that the precision of error detection mechanisms has on ESFL-MAS was investigated. Based on the empirical results, Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support coefficients excel by showing stability while varying the number of passed and failed time steps as well as by reaching high diagnostic accuracy for low error detection precision.

This chapter makes the following contributions:

1. We present an experimental study on the impact of 42 heuristics in the ESFL-MAS diagnostic accuracy using the well-known and real-world representative Pickup and Delivery Problem as test suite;

2. We show that for ESFL-MAS the Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support outperform the remainder coefficients across the entire quantity and quality data space (yielding 96.26% diagnostic accuracy) in the specific conditions of our test suite.

The remainder of this chapter is organised as follows. Section 4.1 explains both the created test suite and the data collection process. In Section 4.2, we evaluate the effects of different similarity coefficients in the diagnosis through varying the amount of available data and the precision of the error detection phase that precedes ESFL-MAS. Finally, Section 4.3 summarises the main topics of this chapter.

4.1 Experimental Setup

The empirical assessment herein presented aims to discover the set of similarity coefficients that yields the best results in terms of diagnostic quality. This section describes the test suite used in the experiments, the data extraction process, and the metric used to evaluate the ESFL-MAS performance.

4.1.1 Test Suite

We use an instance of the Pick-up and Delivery Problem (PDP) [Savelsbergh and Sol, 1995] to test our approach because (i) it is well-known and (ii) it is a real-world representative problem. Problems of this kind are highly dynamic, and decisions have to be made under uncertainty and incomplete knowledge. Briefly, it consists of mobile vehicles retrieving and delivering a set of items. The PDP is a well-studied, NP-hard problem and, given its inherent distribution and decentralisation, MASs offer an interesting solution for PDP [Fischer et al., 1996]. Examples of PDP solved through agents include ride-sharing services [Agatz et al., 2012], application of mobile robots in missions [Posadas et al., 2008] and automated guided vehicle employed in many industrial environments [Grunow et al., 2005].

The Second Edition of the Multi-Agent Programming Contest (MAPC)[1] [Dastani et al., 2007] provides an instance of PDP known as the GoldMiners scenario in which its multiple agents work under uncertainty and responsive situations, i.e., new destinations become available in real-time and are immediately eligible for consideration. GoldMiners implements fundamental concepts of MASs, such as autonomy, team-work coordination, high-level interaction, as well as partial and local perception of the environment. Another reason to use the MAPC's implementation of GoldMiners is that all components composing the system (which include world model, agents' reasoning, and interaction protocols) was previously tested and validated by the MAS community. For our purposes, this assure (in a higher degree of confidence) that only our injected faults will contribute to dysfunctional behaviours.

[1] The MAPC is an annual competition that, since 2005, aims at providing suitable test suite offering key problems for the community to test agent-oriented programming approaches.

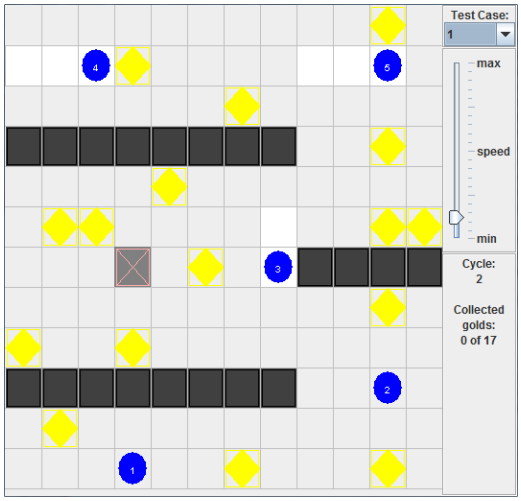


Figure 4.1: Jason's view of the MAPC Goldminers scenario (screenshot).

From several MASs available in the MAPC, we chose the one programmed in AgentSpeak [Rao, 1996], an agent-oriented programming language, while agents were run using Jason [Rafael H. Bordini, 2007], an interpreter for an extended version of AgentSpeak. The choice was made given our previous experience using AgentSpeak [Rossetti et al., 2002, Rossetti and Liu, 2005b,c,a] and the fact that the Jason team won the Second Edition of MAPC.

The MAS aims to solve the PDP by finding a schedule that delivers as many items as possible at the lowest cost while cooperating in a dynamic environment. The environment is a grid-like world where agents can move to a neighbour cell. Agents explore the environment avoiding obstacles and collecting gold nuggets. They also can communicate and coordinate their actions in order to collect as much gold nuggets as possible and to deliver them to the depot where they can be safely stored. In addition, agents have only a local view of the environment, their perceptions might be incomplete, and the executed action may fail.

Research on MASs encompasses different aspects of their artefacts (agents, environment, and so forth) and functioning (organisation, type of social behaviour, knowledge representation, and so forth). As aforementioned, ESFL-MAS is able to diagnose the faulty agent regardless its architecture and knowledge representation. Nevertheless, MAS organisation and type of social behaviour (mainly cooperation) may impact on ESFL-MAS performance. Firstly, when cooperation exists, agents work in high synergy which might contribute to very similar choices influencing the performance spectrum. Secondly, when there is no organisation, agents do not have strict roles in the MAS and so agents' choices might more easily jeopardise another agent performance.

Hence, seeking completeness of the test suite and based on previous work in MAS organisations [Stone and Veloso, 2000, Horling and Lesser, 2004], we implement a modified version of the Jason implementation of MAPC's GoldMiners. Specifically, the original Jason implementation relied on a twofold strategy: first, *a priori* allocation of agents' search quadrants and, second, a team-work coordination aiming to find and carry gold nuggets to the depot. Modified MASs vary both in the *coordination* and in *spatial organisation* (resource allocation) dimensions resulting in the following types of MAS:

1. *Non-Coordinated and Non-Organised* (NCNO): Agents work individually (not cooperatively) and do not receive a search quadrant (loose spatial organization);

2. *Non-Coordinated and Organised* (NCO): Agents work individually but each of them has an assigned search quadrant;

3. *Coordinated and Non-Organised* (CNO): Agents coordinate the gold-nugget search, yet there is no allocated quadrant.

4. *Coordinated and Organised* (CO): Agents coordinate the gold-nugget search as well as have an assigned search quadrant.

As modules responsible for interaction among agents and organisational composition are

Table 4.1: Description of type of faults - Highlighted rows represent the hand-seeded faulty versions and the others are generated through mutation operators.

Qnt.	Fault Description	NCNO	NCO	CNO	CO
1	Agent does not respect its search quadrant	×	√	×	√
1	Agent does not communicate gold-nugget positions	×	×	√	√
1	Agent has a delayed response to an event	√	√	√	√
1	Agent gets stuck in a specific goal	√	√	√	√
1	Agent gets the farthest gold nuggets.	√	√	√	√
3	Delete a belief in the agent.	√	√	√	√
3	Delete a plan in the agent.	√	√	√	√
3	Delete the condition part of a rule.	√	√	√	√
3	Replace the triggering event operator of a plan by another operator.	√	√	√	√
3	Delete the context of a plan if it is non-empty or not set true.	√	√	√	√
3	Delete the body of a plan if it is non-empty or not set true.	√	√	√	√
3	Delete a formula in the body of a non-empty plan.	√	√	√	√
3	Swap the order of any two adjacent formulae in the body of a plan that contains more than one formula.	√	√	√	√
3	Replace the operator of an achievement goal or a mental note formulas by another one.	√	√	√	√
3	Replace the subset of receivers in a communication action.	×	×	√	√
3	Replace the illocutionary force in an action for sending messages by another one.	×	×	√	√
3	Delete a propositional content in the message content.	×	×	√	√

independent from the agent reasoning and other MAS functions, the modified versions inherit the reliability from the original implementation.

Given these baseline (correct) versions, simulating real faults offer a more direct way to assess the proposed technique. These faults can be hand-seeded or seeded by mutation through rules (called mutation operators). We used both of these strategies for different purposes. Hand-seeded faults aim to emulate dysfunctional behaviours specifically for the aforementioned strategy implemented by the Jason Team and, moreover, faults seeded by mutation rules automatically build a set of validated faulty versions as we have used mutation operators proposed by Huang et al. [2014]. In this work we have used the (as called by the authors) high-level mutation operators for Jason.

Table 4.1 gives an overview of the faulty versions in the test suite. Each of these faulty versions contains a single injected fault. In the test suite we were not able to use all created faults for all types of MASs. For instance, Fault 2 cannot be applied to non-cooperative MASs because they inherently do not broadcast any gold nugget positions. Thus, signs $\surd$ and $\times$ in Table 4.1 represent whether a fault has been injected ($\surd$) or not ($\times$).

This test suite covers homogeneous and closed MASs in which agents co-habit in an uncertain environment and may organise to optimally allocate resources and/or interact to establish a team cooperation. Several MASs with such features have been used to solve real-world problems such as those aforementioned in the MASs for PDPs [Agatz et al., 2012, Posadas et al., 2008, Grunow et al., 2005] and we can add traffic control [Passos and Rossetti, 2010b] and shop-floor management [Leitão et al., 2012].

4.1.2 Data Acquisition

A two-step process (shown in Figure 4.2) generates the spectra required by the experiments. In the first stage, we collect simulation logs while in the second we train an error detection mechanism and use such simulation logs to generate the required performance spectra.

Collecting Logs A MAS initially configured according to a given test case is executed to obtain logs from both agents and the overall system. These logs were collected for every test case and they contained the amount of gold nuggets carried by each agent and the total amount of nuggets in the depot for each time step of a simulation.

For the experimental setup, we randomly generated 5 test cases and each of them corresponded to a set of initial positions for: all agents, the depot, and all 400 gold nuggets. Aiming to collect information to generate spectra, the MAS (composed by 25 agents) was executed 75 times for each test case during 1000 time steps. Therefore, we built in total 53,250 performance spectra each with 1000×25 dimension, including every seeded fault and MAS version.

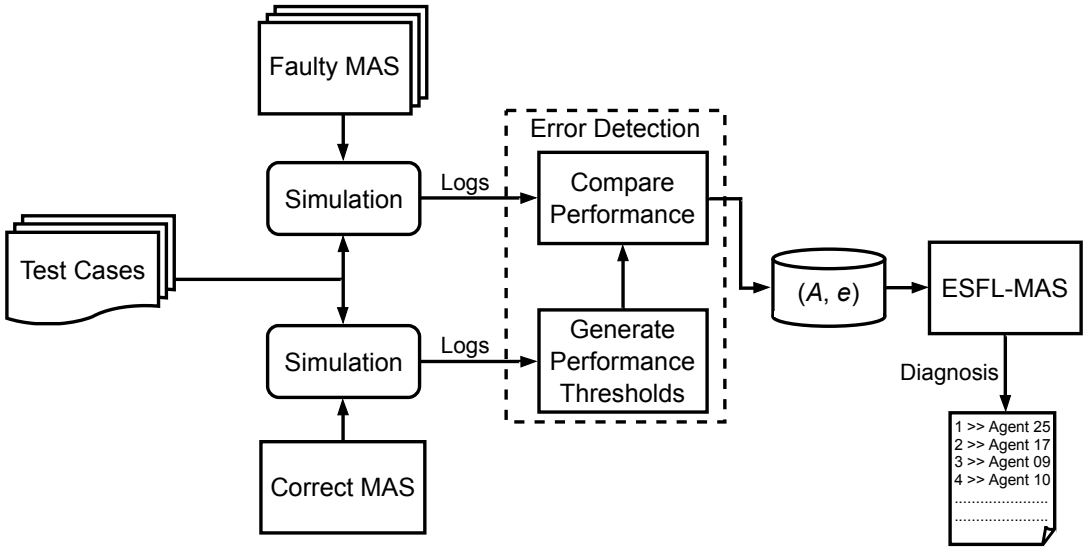


Figure 4.2: Experimental Phases

Expected MAS Performance and Error Detection Both test cases and performance spectra demand for measurements to verify the overall MAS and agent correctness. Agents and the MAS are respectively measured by (1) the amount of carried gold nuggets in each time frame and (2) the amount of gold nuggets in the depot. However, we still need to define the expected MAS performance and error detection mechanism.

As Goldminers can be seen as an PDP instance, the MAS expected output is the average number of gold nuggets in the depot for each time frame. We run the MAS correct version and compute the performance baseline. While assessing faulty versions, time steps with performance values above the baseline are marked as passed ($e_n = 0$) and below as failed ($e_n = 1$). Formally, the system assessment have the underlying pass and fail baseline defined as:

$$e_n = \begin{cases} 1, & \text{if } \#d_n < \Phi(n) \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where $\#d_n$ is the number of gold nuggets in the depot at time n and $\Phi(n)$ is the baseline inferred from the MAS correct version, which can be seen as the expected output of the MAS.

The baseline of a test case is given by:

$$\Phi(n) = \frac{\#d_{max}}{n_{max}} \cdot n \tag{4.2}$$

where $\#d_{max}$ is the maximum number of gold nuggets in the depot and n_{max} is the maximum length of time that simulation had run.

An error detector for agents is also necessary to generate the performance spectra. In this manner, we emulated an error detection phase in our experiments to assess ESFL-MAS, even though these mechanisms are not within the scope of this work. Error detection for Miner agents is calculated similarly to MAS expected performance. We compute the average amount of gold nuggets carried by each agent in a certain time frame and used this value as baseline to detect whether the agent is performing as expected ($A_{nm} = 0$) or not ($A_{nm} = 1$) for time n, therefore mapping collected logs to performance spectra. Formally, the agent error detection have the underlying pass and fail baseline defined as:

$$A_{nm} = \begin{cases} 1, & \text{if } \#Ag_{m_n} < \Psi(n, m) \\ 0, & \text{otherwise} \end{cases} \tag{4.3}$$

where $\#Ag_{m_n}$ is the number of gold nuggets carried by Ag_m at time n and $\Psi(n, m)$ is the baseline inferred from the Ag_m correct version, which can be seen as the nominal behaviour of the agent.

The baseline of an agent is given by:

$$\Psi(n, m) = \frac{\#Ag_{m_{total}}}{n_{max}} \cdot n \tag{4.4}$$

where $\#Ag_{m_{total}}$ is the total number of gold nuggets carried by Ag_m and n_{max} is the maximum length of time that simulation had run.

4.1.3 Evaluation Metric

As ESFL-MAS returns a list of agents sorted by their suspiciousness values (see Definition 10), diagnostic performance is expressed in terms of diagnostic quality (also referred as accuracy) that evaluates how many agents need to be inspected before the faulty agent is found. If other agents have the same similarity coefficient as the faulty agent, we use the average ranking position for these agents. Diagnostic quality is defined as [Steimann et al., 2013]

$$Q = \left(1 - \frac{|\{j|S_j > S_f\}| + |\{j|S_j \geq S_f\}| - 1}{2(M-1)}\right) * 100\% \tag{4.5}$$

where S_j and S_f denote the suspiciousness value for agent j and for the faulty agent respectively, and M is the total number of agents in the system. Intuitively, the $|\{j|S_j > S_f\}|$ term represents the number of agents ranked in front of the faulty agent whereas $|\{j|S_j \geq S_f\}|$ represents the number of agents with same or higher than the suspiciousness value of the faulty one. This metric assumes that, on average, half of agents with same suspiciousness values will be inspected until reaching the fault.

Table 4.2: Example diagnostic report.

Rank	1	2	2	3	4
Agent	Ag_5	Ag_1	Ag_2	Ag_4	Ag_3

As an example consider the diagnostic report presented in Table 4.2 for which the correct diagnostic candidate (i.e. the faulty agent) is Ag_2. In order to calculate the diagnostic quality, we start by examining Ag_5 finding that it is healthy. Due Ag_5 being healthy, we pass to the next agent (Ag_1) in the diagnostic report and it is also healthy. Examining Ag_2 we observe that it is the faulty agent. However, as Ag_1 is tied with Ag_2, we must inspect half of the agents. The terms $|\{j|S_j > S_{Ag_2}\}| = |\{Ag_5\}| = 1$ and $|\{j|S_j \geq S_{Ag_2}\}| = |\{Ag_5, Ag_1, Ag_2\}| = 3$. The diagnostic quality is therefore:

$$Q = \left(1 - \frac{1+3-1}{2(5-1)}\right) * 100\% = \left(1 - \frac{3}{8}\right) * 100\% = 62.5\% \tag{4.6}$$

4.1.4 List of Similarity Coefficients

In this chapter, an exhaustive list of 42 heuristics (a.k.a. similarity coefficients) [Hofer et al., 2015, Lucia et al., 2014] has been studied, focusing on the context of fault localisation in software agents. These coefficients are: Accuracy (C_1), Added Value (C_2), Anderberg (C_3), Certainty Factor (C_4), Collective Strength (C_5), Confidence (C_6), Conviction (C_7), Coverage (C_8), Example and Counterexample (C_9), Gini Index (C_{10}), Goodman and Kruskal (C_{11}), Information Gain (C_{12}), Interest (C_{13}), Interestingness Weighting Dependency (C_{14}), J-Measure (C_{15}), Jaccard (C_{16}), Kappa (C_{17}), Klosgen (C_{18}), Laplace (C_{19}), Least Contradiction (C_{20}), Leverage (C_{21}), Loevinger (C_{22}), Normalized Mutual Information (C_{23}), Ochiai (C_{24}), Ochiai II (C_{25}), Odd Multiplier (C_{26}), Odds Ratio (C_{27}), One-way Support (C_{28}), Piatetsky-Shapiro (C_{29}), Relative Risk (C_{30}), Rogers and Tanimoto (C_{31}), Sebag-Schoenauer (C_{32}), Simple-Matching (C_{33}), Sorensen-Dice (C_{34}), Support (C_{35}), Tarantula (C_{36}), Two-way Support (C_{37}), Two-way Support Variation (C_{38}), Yule's Q (C_{39}), Yule's Y (C_{40}), Zhang (C_{41}), ϕ-coefficient (C_{42}). The mathematical formulae for computing each coefficient are shown in Table 4.3.

In this context, the variable A and B are the two dimensions of the dichotomy matrix in Table 3.4, corresponding to healthy state of the agent and status of the MAS performance, respectively. The table also uses standard notation from probability and statistics, namely: $P(A)$ is the probability of A, $P(\overline{A})$ is the probability of *not* A, $P(AB)$ is the joint probability of A and B. ESFL-MAS gathers frequencies rather then probabilities, but these can be easily substituted during actual computation (see Table 4.4).

Table 4.3: Similarity Coefficients and their formulae.

#	**Coefficient**	**Formulae**
C_1	Accuracy	$P(AB) + P(\overline{A}\overline{B})$
C_2	Added Value	$max\left(P\left(B\middle\vert A\right) - P\left(B\right), P\left(A\middle\vert B\right) - P\left(A\right)\right)$
C_3	Anderberg	$\frac{P(AB)}{P(AB) + 2(P(AB) + P(A\overline{B}))}$
C_4	Certainty Factor	$max\left(\frac{P\left(B\middle\vert A\right) - P\left(B\right)}{1 - P\left(B\right)}, \frac{P\left(A\middle\vert B\right) - P\left(A\right)}{1 - P\left(A\right)}\right)$
C_5	Collective Strength	$\frac{P(AB) + P(\overline{A}\overline{B})}{P\left(A\right)P\left(B\right) + P(\overline{A})P(\overline{B})} \times \frac{1 - P\left(A\right)P\left(B\right) - P(\overline{A})P(\overline{B})}{1 - P\left(AB\right) - P(\overline{A}\overline{B})}$
C_6	Confidence	$max\left(P\left(B\middle\vert A\right), P\left(A\middle\vert B\right)\right)$
C_7	Conviction	$max\left(\frac{P\left(A\right)P(\overline{B})}{P(A\overline{B})}, \frac{P\left(B\right)P(\overline{A})}{P(B\overline{A})}\right)$
C_8	Coverage	$P\left(A\right)$
C_9	Example and Counterexample	$1 - \frac{P(A\overline{B})}{P(AB)}$
C_{10}	Gini Index	$max(P\left(A\right)\left[P\left(B\middle\vert A\right)^2 + P(\overline{B}\middle\vert A)^2\right] + P(\overline{A})[$ $P(B\middle\vert\overline{A})^2 + P(\overline{B}\middle\vert\overline{A})^2] - P\left(B\right)^2 - P(\overline{B})^2,$ $P\left(B\right)\left[P\left(A\middle\vert B\right)^2 + P(\overline{A}\middle\vert B)^2\right] + P(\overline{B})[P(A\middle\vert\overline{B})^2 +$ $+P(\overline{A}\middle\vert\overline{B})^2] - P\left(A\right)^2 - P(\overline{A})^2)$
C_{11}	Goodman and Kruskal	$\frac{\sum_i max_j P\left(A_i B_j\right) + \sum_j max_i P\left(A_i B_j\right) - max_i P\left(A_i\right) - max_j P\left(B_j\right)}{2 - max_i P\left(A_i\right) - max_j P\left(B_j\right)}$
C_{12}	Information Gain	$(-P(B)logP(B) - P(\overline{B})logP(\overline{B}))-$ $-(P(A) \times (-P(B\middle\vert A)logP(B\middle\vert A))-$ $-P(\overline{B}\middle\vert A)logP(\overline{B}\middle\vert A) - P(\overline{A})\times$ $\times(-P(B\middle\vert\overline{A})logP(B\middle\vert\overline{A})) - P(\overline{B}\middle\vert\overline{A})logP(\overline{B}\middle\vert\overline{A}))$
C_{13}	Interest	$\frac{P(AB)}{P\left(A\right)P\left(B\right)}$
C_{14}	Interestingness Weighting Dependency	$\left(\left(\frac{P(AB)}{P\left(A\right)P\left(B\right)}\right)^k - 1\right) P\left(AB\right)^m$,where k, m are coefficients of dependency and generality respectively
C_{15}	J-Measure	$max(P(AB)log\left(^{P(B\middle\vert A)}/_{P(B)}\right) + P(A\overline{B})log(^{P(\overline{B}\middle\vert A)}/_{P(\overline{B})}),$ $P(AB)log\left(^{P(A\middle\vert B)}/_{P(A)}\right) + P(\overline{A}B)log(^{P(\overline{A}\middle\vert B)}/_{P(\overline{A})}))$
C_{16}	Jaccard	$\frac{P(AB)}{P\left(A\right) - P\left(B\right) - P\left(AB\right)}$
C_{17}	Kappa	$\frac{P(AB) + P(\overline{A}\overline{B}) - P\left(A\right)P\left(B\right) - P(\overline{A})P(\overline{B})}{1 - P\left(A\right)P\left(B\right) - P(\overline{A})P(\overline{B})}$
C_{18}	Klosgen	$\sqrt{P\left(AB\right)}max\left(P\left(B\middle\vert A\right) - P\left(B\right), P\left(A\middle\vert B\right) - P\left(A\right)\right)$
C_{19}	Laplace	$max\left(\frac{P\left(AB\right)+1}{P\left(A\right)+2}, \frac{P\left(AB\right)+1}{P\left(B\right)+2}\right)$
C_{20}	Least Contradiction	$\frac{P(AB) - P(A\overline{B})}{P\left(B\right)}$
C_{21}	Leverage	$P\left(B\middle\vert A\right) - P\left(A\right)P\left(B\right)$
C_{22}	Loevinger	$1 - \frac{P\left(A\right)P(\overline{B})}{P(A\overline{B})}$
C_{23}	Normalized Mutual Information	$\left(\sum_i \sum_j P(A_i B_j) log_2 \frac{P(A_i B_j)}{P(A_i)P(B_j)}\right) / \left(-\sum_i P(A_i) log_2 P(A_i)\right)$
C_{24}	Ochiai	$\frac{P(AB)}{\sqrt{P\left(A\right)P\left(B\right)}}$
C_{25}	Ochiai II	$\frac{P(AB) + P(\overline{A}\overline{B})}{\sqrt{(P(AB) + P(\overline{A}B))(P(AB) + P(A\overline{B}))(P(\overline{A}\overline{B}) + P(\overline{A}B))(P(\overline{A}\overline{B}) + P(A\overline{B}))}}$

Continued on next page.

Continued from previous page.

#	Coefficient	Formulae
C_{26}	Odd Multiplier	$\frac{P(AB)\,P(\overline{B})}{P(B)\,P(A\overline{B})}$
C_{27}	Odds Ratio	$\frac{P(AB)\,P(\overline{AB})}{P(A\overline{B})P(\overline{A}B)}$
C_{28}	One-way Support	$P(B\|A)\,log_2\frac{P(AB)}{P(A)\,P(B)}$
C_{29}	Piatetsky-Shapiro	$P(AB) - P(A)\,P(B)$
C_{30}	Relative Risk	$\frac{P(B\|A)}{P(B\|\overline{A})}$
C_{31}	Rogers and Tanimoto	$\frac{P(AB) + P(\overline{AB})}{P(AB) + P(\overline{AB}) + 2(P(\overline{A}B) + P(A\overline{B}))}$
C_{32}	Sebag-Schoenauer	$\frac{P(AB)}{P(A\overline{B})}$
C_{33}	Simple-Matching	$P(AB) + P(\overline{AB})$
C_{34}	Sorensen-Dice	$\frac{2P(AB)}{2P(AB) + P(\overline{A}B) + P(A\overline{B})}$
C_{35}	Support	$P(AB)$
C_{36}	Tarantula	$\frac{P(AB)/P(B)}{P(A\overline{B})/P(\overline{B}) + P(AB)/P(B)}$
C_{37}	Two-way Support	$P(AB)\,log_2\frac{P(AB)}{P(A)\,P(B)}$
C_{38}	Two-way Support Variation	$P(AB)log_2\frac{P(AB)}{P(A)P(B)} + P(\overline{A}B)log_2\frac{P(\overline{A}B)}{P(\overline{A})P(B)} + +P(A\overline{B})log_2\frac{P(A\overline{B})}{P(A)P(\overline{B})} + P(\overline{AB})log_2\frac{P(\overline{AB})}{P(\overline{A})P(\overline{B})}$
C_{39}	Yule's Q	$\frac{P(AB)\,P(\overline{AB}) - P(A\overline{B})P(\overline{A}B)}{P(AB)\,P(\overline{AB}) + P(A\overline{B})P(\overline{A}B)}$
C_{40}	Yule's Y	$\frac{\sqrt{P(AB)\,P(\overline{AB})} - \sqrt{P(A\overline{B})P(\overline{A}B)}}{\sqrt{P(AB)\,P(\overline{AB})} + \sqrt{P(A\overline{B})P(\overline{A}B)}}$
C_{41}	Zhang	$\frac{P(AB) - P(A)\,P(B)}{max(P(AB)\,P(\overline{B}), P(B)\,P(A\overline{B}))}$
C_{42}	ϕ-coefficient	$\frac{P(AB) - P(A)\,P(B)}{\sqrt{P(A)\,P(B)\,(1-P(A))\,(1-P(B))}}$

Table 4.4: Definitions of the probabilities used by coefficients in the context of this chapter.

$$P(A) = \frac{c_{11} + c_{10}}{c_{11} + c_{10} + c_{01} + c_{00}} \qquad P(\overline{A}) = \frac{c_{01} + c_{00}}{c_{11} + c_{10} + c_{01} + c_{00}}$$

$$P(B) = \frac{c_{11} + c_{01}}{c_{11} + c_{10} + c_{01} + c_{00}} \qquad P(\overline{B}) = \frac{c_{10} + c_{00}}{c_{11} + c_{10} + c_{01} + c_{00}}$$

$$P(AB) = \frac{c_{11}}{c_{11} + c_{10} + c_{01} + c_{00}} \qquad P(\overline{A}B) = \frac{c_{01}}{c_{11} + c_{10} + c_{01} + c_{00}}$$

$$P(A\overline{B}) = \frac{c_{10}}{c_{11} + c_{10} + c_{01} + c_{00}} \qquad P(\overline{AB}) = \frac{c_{00}}{c_{11} + c_{10} + c_{01} + c_{00}}$$

$$P(A|B) = \frac{P(AB)}{P(B)} \qquad P(B|A) = \frac{P(AB)}{P(A)}$$

4.2 Experimental Results

As this study investigates an exhaustive list of similarity coefficients and some of them behave very similarly to others, we decided to perform a clustering analysis. Hierarchical clustering was used to construct the dendrogram [McQuitty, 1966], together with bootstrapping analysis as no assumption about the diagnostic accuracy distribution could be made. Several hierarchical clustering approaches (with different linkage and distances metrics) yielded similar dendrograms. However, the results discussed in this chapter were obtained using the weighted average as linkage [McQuitty, 1966] and the uncentered Pearson correlation as distance [Pearson, 1895], because they generate statistically significant results with marginal standard error (SE). In order to select the number of clusters by cutting the generated tree, we used the Approximately Unbiased test [Shimodaira, 2004], which corrects the bias caused by bootstrap, with 95% confidence (0.001 SE). In this way, coefficients within the same cluster present low inter-cluster distances and high intra-cluster distances.

The resulting groups for the NCNO, NCO, CNO, and CO types of MAS are shown in Figures 4.3.a, 4.3.b, 4.3.c, and 4.3.d, respectively. Each circle corresponds to a cluster (a.k.a. group), and edges indicate relationships between clusters such that A $\longrightarrow$ B means "group A requires less effort to diagnose than group B". The number of groups (and their elements) is not the same in every type of MAS. There are 7 groups for NCNO and NCO whereas 5 are the groups for CNO and CO. The non-coordinated versions tend to have more disperse results than coordinated versions; this occurs because non-coordinated agents rely solely on their own capability to perceive the environment, where coordinated agents have an expanded perception through their peers. Furthermore, some clusters yield similar diagnostic quality, but they are statistically different, that is, they generate different diagnostic reports. These clusters appear horizontally aligned in Figure 4.3.

After carefully analysing the clusters, two fundamental aspects can explain their composition. First, from the mathematical formulae of coefficients, there is a logical causal relationship between accuracy and the dichotomy matrix: the more significance a coefficient

Table 4.5: Mean accuracy for each similarity coefficient.

Group	Diagnostic Quality ($\overline{Q}\,(\sigma)\,[\%]$)			
	NCNO	NCO	CNO	CO
01	96.25 (11.27)	95.16 (12.68)	97.08 (10.35)	96.54 (10.74)
02	73.96 (31.15)	67.74 (27.45)	55.83 (39.88)	66.23 (36.46)
03	53.61 (22.98)	53.27 (27.02)	54.10 (31.77)	49.95 (1.628)
04	47.90 (12.90)	47.87 (17.81)	50.00 (0.00)	44.92 (21.33)
05	43.54 (19.61)	46.19 (14.26)	8.358 (19.67)	27.36 (38.17)
06	36.85 (21.58)	37.47 (21.39)	-	-
07	22.08 (27.95)	23.32 (31.46)	-	-

assigns to anomalous behaviour for which a system's error has been detected (represented by c_{11}), the better the diagnostic quality. Second, the test suite has reduced capacity of representing cascading effects of undesired behaviour; for instance, the faulty agent sending the wrong location of a gold nugget for a correct agent inducing the latter in failure. Accordingly, coefficients that give greater emphasis on time steps where MAS was correct and an agent had an unexpected behaviour have lower diagnostic quality.

The overall mean diagnostic quality for the test suite, along with the standard deviation (σ), are presented in Table 4.5. Each cell in the table corresponds to a cluster in Figure 4.3; for instance, Group NCO-03 is in the column NCO and row 03 of the table. Groups NCNO-01, NCO-01, CNO-01, and CO-01 yield the best diagnostic accuracy with 96.25%(11.27%), 95.16%(12.68%), 97.08%(10.35%), and 96.54%(10.74%), respectively. This means that the user have to inspect approximately 4% of the agents to find the faulty one. These groups also show low standard deviation (σ), excluding Groups CNO-04 and CO-03. Actually, the results of Groups CNO-04 and CO-03 are masked because similarity coefficients within these groups compute the same suspiciousness values for every agent in the diagnostic report and, as assumed by the evaluation metric, the faulty agent is in the middle of the list, resulting in the approximate accuracy of 50% and standard deviation 0%. In practice, these results are not useful and are not considered in our analysis.

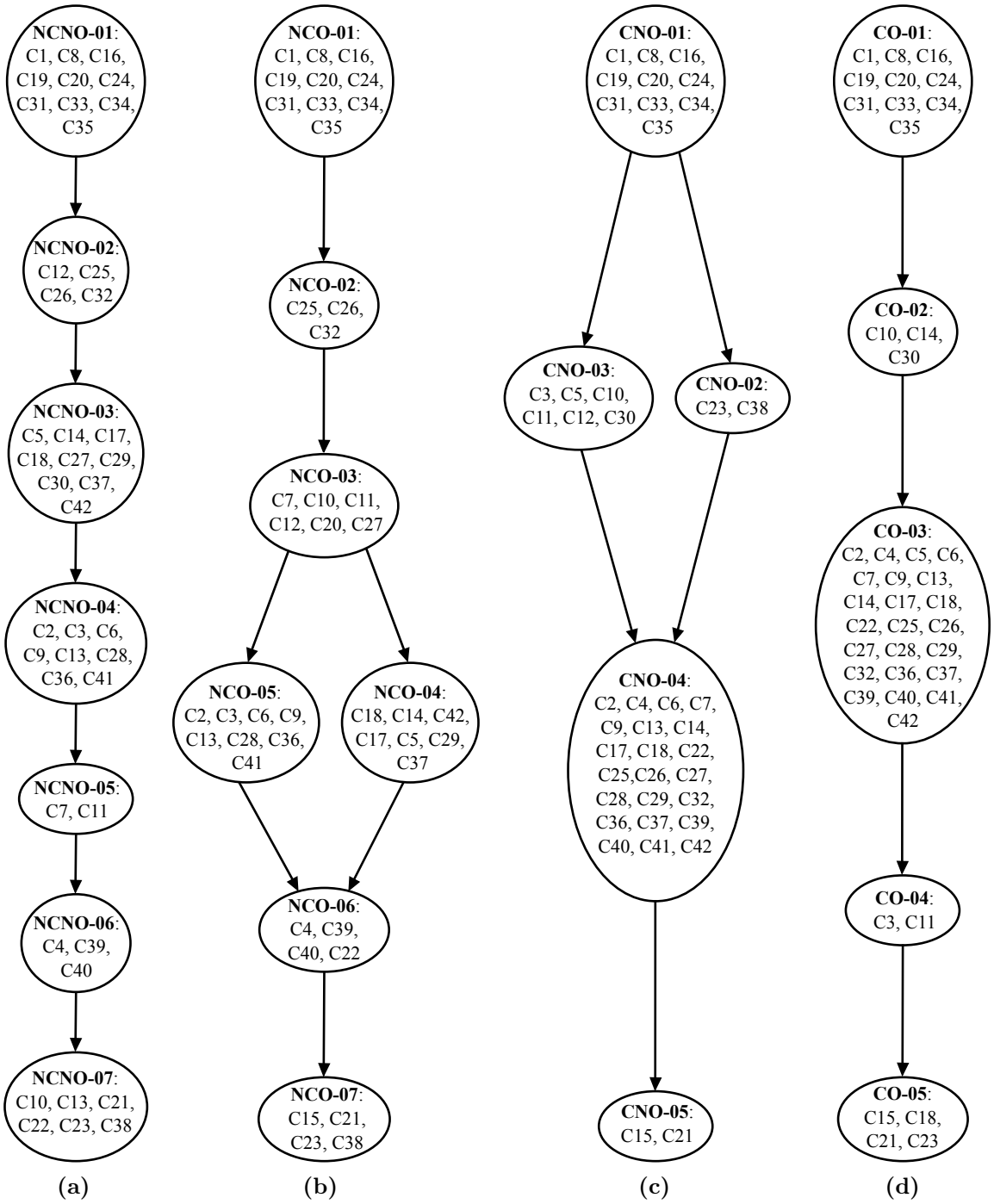


Figure 4.3: Similarity coefficients grouped by their quality of diagnosis: each node corresponds to a group; edges indicate relationships between groups such that A → B means "group A requires less effort to diagnose than group B"; those with the same horizontal alignment present less than 1% difference in the mean quality of diagnosis.

4.2.1 On the Impact of Observation Quantity

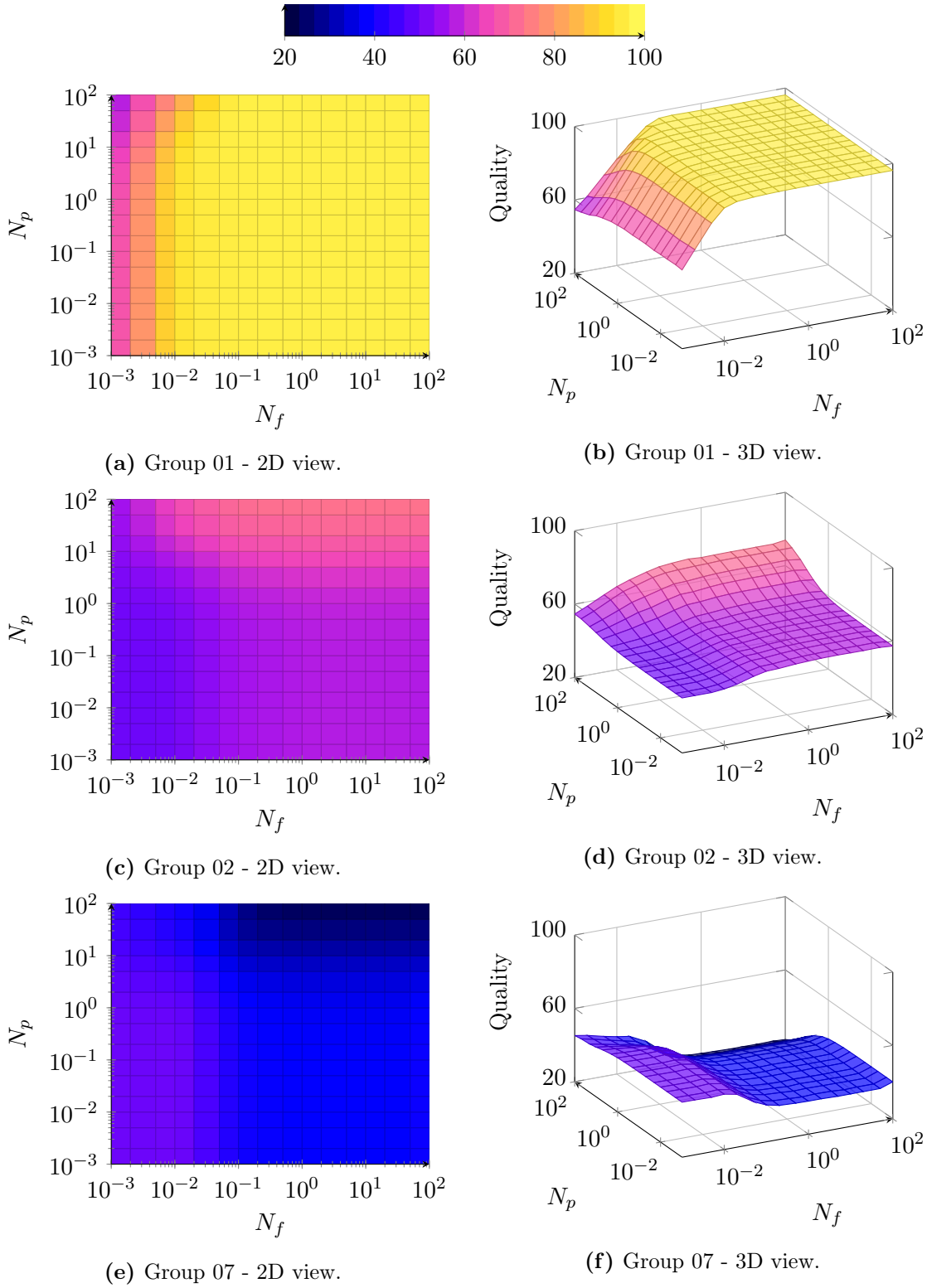
In the previous results, we have assumed that there is enough time to run the MAS several rounds under different conditions to collect a considerable amount of measurements. In practice, however, the tester works under short-time constraints or he/she does not want to extensively assess the system. To investigate the influence of the amount of available data on ESFL-MAS performance, we evaluate Q while varying the number of passed (N_p) and failed time steps (N_f) that are available. The spectrum time-line is not rearranged during the variation of both N_p and N_f; the process here is to randomly exclude non-consecutive time steps. We did not want to compromise the consistency of our data, even though ESFL-MAS assumes that time steps are independent. Since the ratio of failed and passed time steps is very small (nearly 0.02) and no previous experiment analysing the ESFL-MAS sensitivity have been performed, we study the influence of the quantity of available data on the diagnostic accuracy Q across the entire range of available data. Thus, N_p and N_f are varied from 0.001% to 100% of the total number of passed and failed time steps and results are presented in logarithmic scale.

Figures 4.4 and 4.5 show such evaluations of Groups NCNO-01, NCNO-02, and NCNO-07, and CO-01, CO-02, and CO-05, respectively. They allow a comparison of the ESFL-MAS behaviour across the two most different MAS versions as well as of groups of the same MAS version. For each graph, we averaged Q over 1000 randomly selected combinations of passed and failed time steps until the variance in the measured values of Q is negligible.

Concerning the number of erroneous time steps N_f, we confirm from Figures 4.4 and 4.5 that adding failed time steps improves the diagnostic quality for most case. The benefit of including more than 200 N_fs for Groups NCNO-01 (Figures 4.4.a and 4.4.b) and CO-01 (Figures 4.5.a and 4.5.b) is marginal on average. This number increases to 2000 N_fs for Groups NCNO-02 (Figures 4.4.c and 4.4.d) and CO-02 (Figures 4.5.c and 4.5.d). Conversely, coefficients in Groups NCNO-07 (Figures 4.4.e and 4.4.f) and CO-05 (Figures 4.5.e and 4.5.f) lose performance when including more failed time steps. This happens once these similarity coefficients are inversely proportional to c_{11} and c_{01}.

Concerning the number of correct time steps N_p, results for each version are quite different. As for CO MASs, N_p did not influence the ESFL-MAS accuracy (see Figure 4.5). This phenomenon can be explained by the presence of coordination among agents: when agents work as team-mates in an organised manner, the MAS performs as a "well-oiled machine" and an agent that fails under these circumstances is more easily detected and therefore more easily correlated with the system failure. As for NCNO MASs, correct time steps can have effects on the diagnostic quality: (1) slightly degrade the ESFL-MAS performance for $N_f < 0.1\%$ (see Figures 4.4.a and 4.4.b); (2) positively influence the ESFL-MAS results across N_f dimension (see Figures 4.4.c and 4.4.d); and (3) decrease ESFL-MAS accuracy for $N_f > 0.1\%$ (see Figures 4.4.e and 4.4.f). These sparse results are consequence of the chaos[2] created by autonomic decisions.

[2]The word chaos refers to the Chaos Theory, which is the area of mathematics that studies the behaviour of dynamic systems that are highly sensitive to initial conditions.



(a) Group 01 - 2D view.

(b) Group 01 - 3D view.

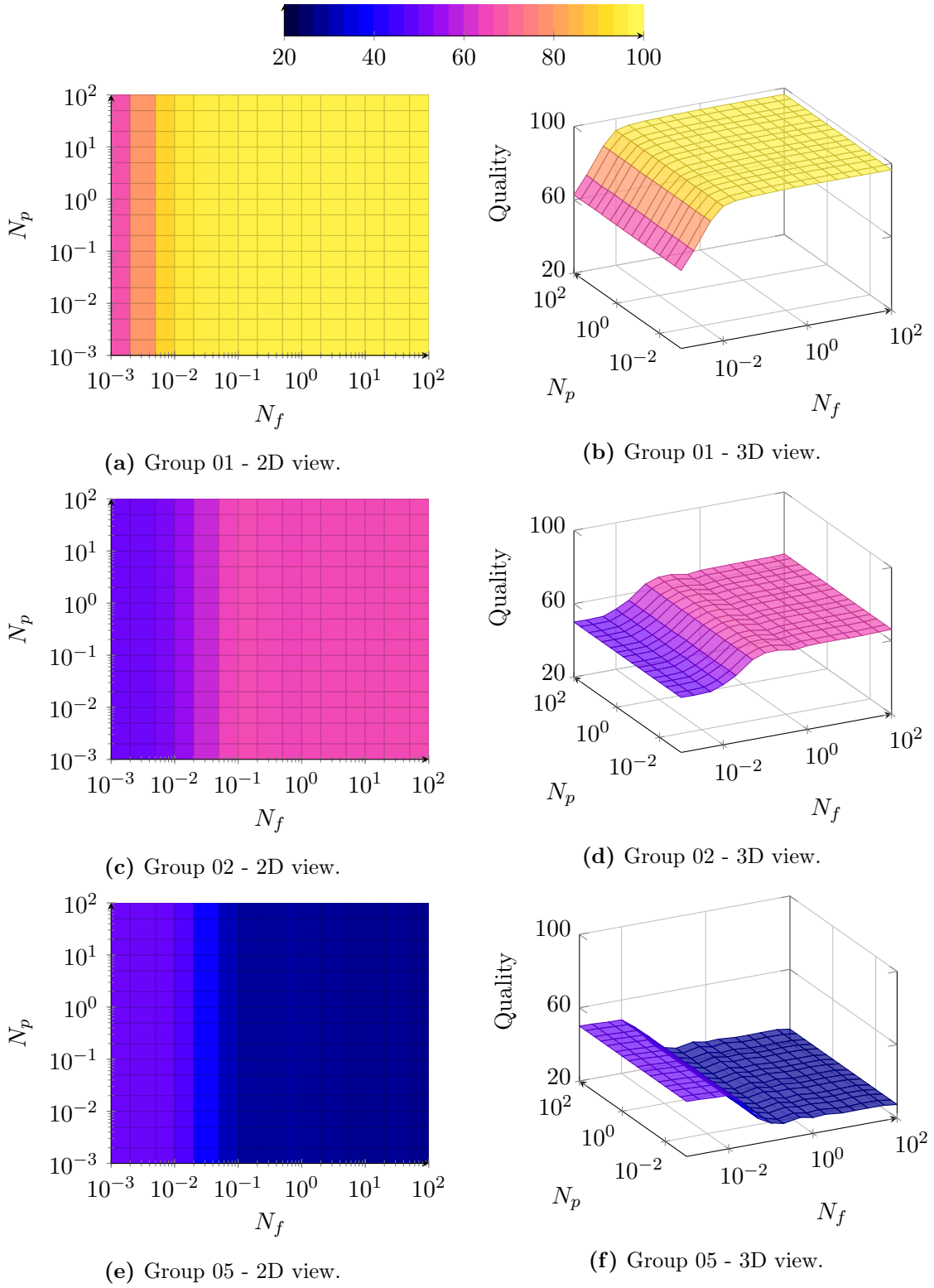
(c) Group 02 - 2D view.

(d) Group 02 - 3D view.

(e) Group 07 - 2D view.

(f) Group 07 - 3D view.

Figure 4.4: Observation quantity impact of NCNO MASs.



(a) Group 01 - 2D view.

(b) Group 01 - 3D view.

(c) Group 02 - 2D view.

(d) Group 02 - 3D view.

(e) Group 05 - 2D view.

(f) Group 05 - 3D view.

Figure 4.5: Observation quantity impact of CO MASs.

4.2.2 On the Impact of Error Detection Precision

Error detection is the phase that precedes diagnosis and as such it has a great impact on the ESFL-MAS' diagnostic quality. The following experiment shows how precision in detecting error affects diagnostic quality for each similarity coefficient. For any realistic system and practical error detection mechanism, there will very likely exist errors that go undetected, mainly because of two reasons. Firstly, the faulty agent only jeopardises system operation under specific scenario settings. For instance, let us assume that a Miner agent, erroneously, is not able to perceive gold nuggets; no error is detected unless the agent gets near a gold nugget. Secondly, analogously to faults in software, errors induced by agents might not propagate all the way to system failures and thus go undetected.

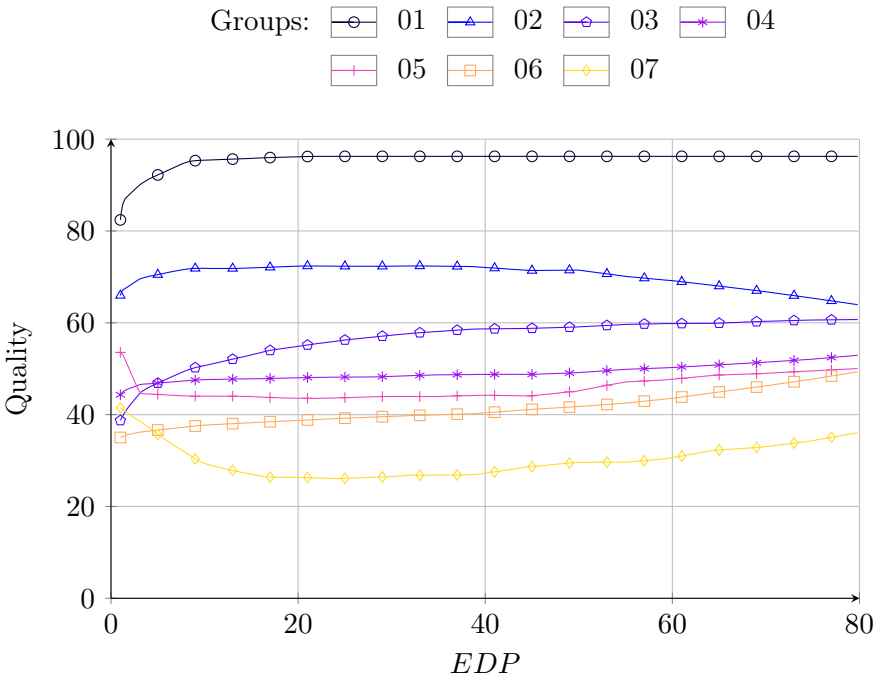
As consequence of these issues, the number of rows in spectra, in which both faulty agent and system fail, will only be a fraction of the total rows in which the agent fails. More intuitively, this proportion represents the Error Detection Precision (EDP), that is, how precisely the error detection phase is able to correlate a system failure with the faulty agent. Using the previous notation, we define

$$EDP = \left(\frac{c_{11}(f)}{c_{11}(f) + c_{10}(f)} \right) * 100\% \tag{4.7}$$

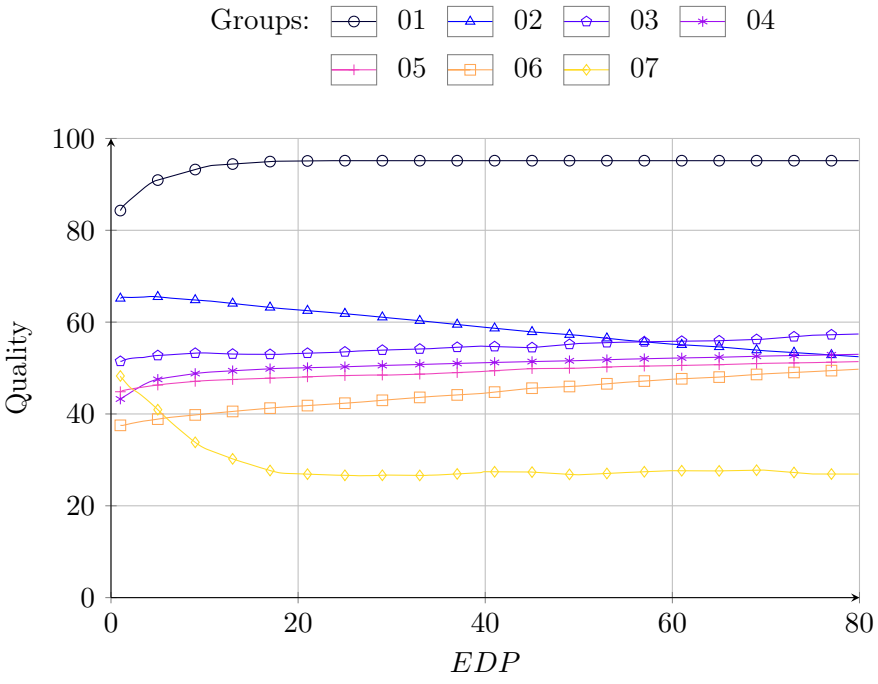
where f is the location of the faulty agent. By varying the EDP ratio, we are assessing the response of the ESFL-MAS diagnostic quality for different error detection mechanisms.

Each faulty version of our test suite has an inherent value for EDP fluctuating from 3.31% to 97.77%. We vary EDP using two methods: (1) excluding time steps that activate the faulty agent, but for which no system error has been detected decreasing $c_{10}(f)$, and *increasing* EDP; and (2) excluding time steps that activate the faulty agent and for each an system error has been detected decreasing $c_{11}(f)$, and *decreasing* EDP. In order to unbias our experiment, we randomly sample passed and failed time steps from the set of available ones to control EDP within a 95% confidence interval. Yet, similarly to the experiment of observation quantity impact (Section 4.2.1), this random selection of passed and failed time steps maintained their sequence as the original spectrum to sustain consistency. These time step exclusions can be seen in practice as incomplete monitoring on agents (instrumentation blindness). The experiments serve well to demonstrate whether or not the approach is robust to such incompleteness.

Figures 4.6 and 4.7 depict how the diagnostic quality changes with respect to the error detection precision for non-coordinated and coordinated versions respectively. One can see that, on average for all cases, a detection precision greater than 40% have marginal contribution to a better fault diagnosis. This does not mean that the community needs to give up improving error detection techniques; this means that, when coupled with a diagnosis phase, error detection needs a solid (but non-optimal) performance. Moreover, we confirm Group 01 as the best set of similarity coefficients for MAS also regarding EDP variation. We show that ESFL-MAS can achieve high accuracy even for low error detection precision being the borderline $EDP \geq 10\%$.

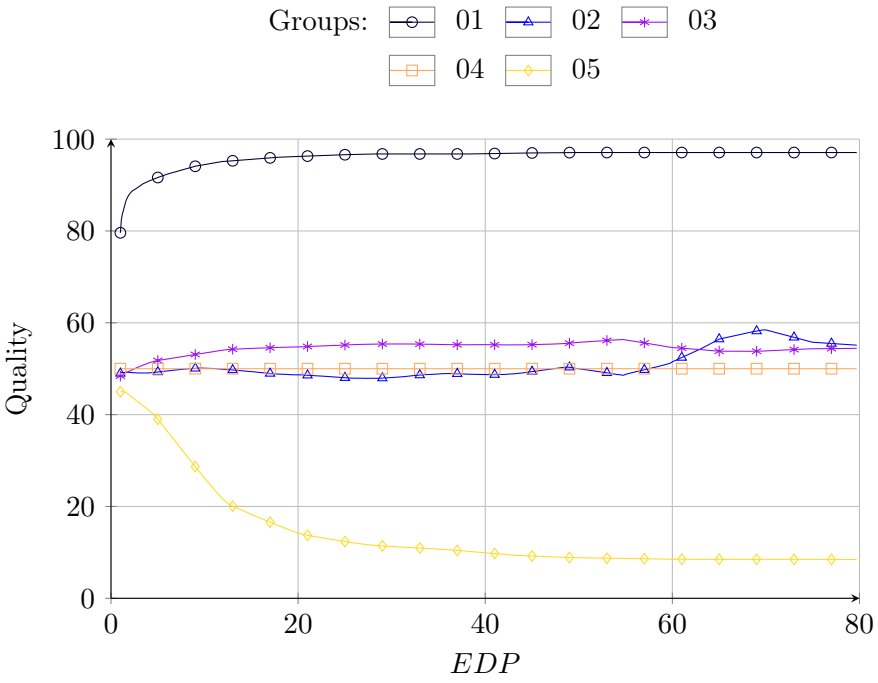


(a) NCNO MAS.

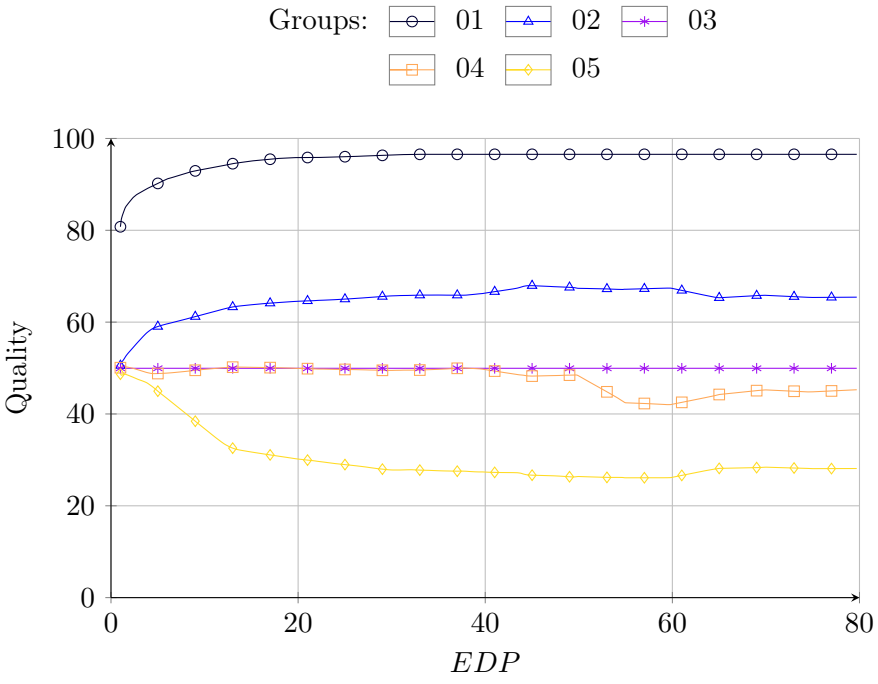


(b) NCO MAS.

Figure 4.6: *EDP* for Non-Coordinated MAS versions.



(a) CNO MAS.



(b) CO MAS.

Figure 4.7: *EDP* for Coordinated MAS versions.

4.2.3 Discussion

From observation quality impact, the N_p effect over ESFL-MAS performance degrades as agents work cooperatively in an organised fashion. Hence, our results suggest that a single faulty agent can more easily be pinpointed in a team organisation (without significant cascading errors) rather than in a selfish MAS. Additionally, adding N_f improves the diagnostic quality until 200 time frames when its contribution starts to become marginal.

Regarding the effect of the error detection precision, ESFL-MAS shows to be robust for a broad range of EDP and Groups NCNO-01, NCO-01, CNO-01, and CO-01 produced the best and more stable performance being their near-optimal response achieved when $EDP \geq 10\%$.

Experiments suggest that ESFL-MAS accuracy might be jeopardised by cascading faults produced by highly interacting agents. Furthermore, our experiments determined that the best similarity coefficients for ESL-MAS, are Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support yield the best results in our experiment.

In agreement with our findings, literature also reports the Ochiai coefficient has yielded the best diagnostic quality when SFL is applied to diagnose faults in practical domains [Hofer et al., 2015, Abreu et al., 2009, Le et al., 2013]. This suggests that Ochiai is a good candidate to be incorporated in a practical implementation of our ESFL-MAS.

Several granularity levels might be considered while diagnosing faults in MASs. This initial proposal of ESFL-MAS localises agent behavioural faults aiming at its on-line usage to increase reliability in the system as a whole. Nevertheless, ESFL-MAS could be applied to a lower granularity level inside an agent, for instance, diagnosing flaws in their rules or plans. This finer granularity may help designers to better debug agents' behaviours and improve their functioning. Without any prior practical knowledge, ESFL-MAS could be employed to this purpose as far as the instrumentation was done at the desired level. Of course that limitations would appear as the research advances, but it is important to highlight that the described technique is a solid starting point for these different abstraction levels of MASs.

Threats to internal validity might come from how the empirical study was carried out. To reduce the probability of having faults in our experimental framework, it has been carefully tested. Furthermore, the random process of selecting spectrum rows are affected by chance. Thus, we repeated each described experiments and applied rigorous statistical procedures to assess results. Although we have chosen a well-known instance of PDP problem that was effectively tested during the MAPC, there are threats to external validity. The main one is the use of only one single scenario, which constrains the generalisation of the obtained results. Even though the injected faults created through mutation operators were randomly chosen and our experiments used a more faulty version that any other efforts in the literature, another threat might come from the limited type of used operators. To allow reproducibility of results, all versions and test cases will be available on a public repository.

4.3 Summary

The performance of ESFL-MAS greatly depends on particular factors, namely: (1) similarity coefficient, (2) quantity of observations, and (3) quality of error detectors. These dependencies have been thoroughly studied by the empirical assessments, which yielded prominent results giving a good prospect for the application of ESFL-MAS. Results show that Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support yield the best diagnostic accuracy for the created test suite. They yield roughly 96.26% diagnostic accuracy and are stable when varying both error detection precision and quantity of observations.

Chapter 5

Conclusion

Several MASs application have been deployed over the past decade in real-world scenarios. They demand for agents capable of assuring nominal behaviour in the presence of unforeseen events on a sound and safe basis. Increasing reliability of distributed, intelligent entities mainly relies on the correct diagnosis of faulty components responsible for MAS failures. Current approaches, proposed specifically to MASs, capable of these tasks like (1) formal verification techniques, (2) testing tools, and (3) fault tolerance frameworks generally assume *a priori* knowledge of the system that describe expected behaviour. Such an assumption may not hold while dealing with complex MASs that operated in dynamic environments with inherent legacy components, because to shape this knowledge in the form of models is a laborious and error-prone work for the designers. Thus, accuracy of fault diagnosis approaches may decrease while using incomplete and/or unreliable model of the system.

In this thesis, we considered that diagnosis techniques for MASs should depend on minimal knowledge about the system to pinpoint behavioural faults of agents. Based on spectrum-based reasoning, the proposed approach, named ESFL-MAS, is able to identify agents that may jeopardise the overall performance through run-time profiles of the system with high diagnostic accuracy. In particular, we (1) studied the limitations of applying SFL to time-persistent and autonomous entities and (2) extend the spectrum-based fault localisation to support MAS features, including a filter to augment useful information within spectra.

Moreover, similarity-based SFL has no standard heuristic that present optimal performance for every domain; therefore, we conducted an empirical evaluation aimed at establishing the best set of similarity coefficients for the MAS context. The built experimental setup was based on the Jason implementation for Goldminers scenario. We were able to expand our ESFL-MAS assessment by developing different strategies of both agent coordination and organisation; additionally, a broad range of fault types (from hand-seeded to mutation operators) was injected to properly represent realistic abnormal behaviours. Experiment consisted of (1) a study of how the quantity of run-time observation impacts the ESFL-MAS performance, and (2) an analysis on the error detection precision so ESFL-MAS is able to sustain a high diagnostic quality.

5.1 Main Contributions

In this section, we will recover the research questions outlined in Section 1.2 and explain the path followed in our research so as to provide answers to these questions resulting in the main contributions of this thesis.

SFL has been studied in different types of software applications [Abreu et al., 2009, Le et al., 2013, Hofer et al., 2015], even diagnosing faults in concurrent blocks of code [Koca et al., 2013]. However, spectrum-based diagnosis has never been experimented in MASs. This is why we have formulated the first research question, which we here recapitulate:

1. *How to encode the time perspective and the autonomy into spectrum-based algorithms as a way of diagnosing behavioural faults in MASs?*

Two steps were followed to answer this question: discover the limitation of SFL related to MAS context and extend SFL to meet our requirements of diagnosing behaviour faults of agent-based systems.

To accomplish the first step, we mapped concepts of MASs to the basic elements of SFL. As this thesis deals with agent-level faults, components were mapped to agents themselves. While trying to deal with transactions, we face the first SFL limitation. By definition, transactions are sets of component information, whose correctness of output can be verified and usually they are encoded as the involvement of each component of the system in terms of involved/not involved in execution of a particular test case. However, since agents are always perceiving and acting upon the environment, we would have an uniform spectrum with no useful information to discover the faulty agent. The second SFL limitation concerns the autonomic facet of the agent. In the traditional SFL, to pinpoint the fault component the algorithm assumes that they have the same structure and equally operate under the same circumstances. This assumption does not hold for MASs as agents might act differently while facing the same events. Therefore, we need to extend SFL to encompass concepts of MASs.

The Extended Spectrum-based Fault Localization for MAS (ESFL-MAS) solves the first limitation by proposing the performance spectra, which encodes the performance of each agent in terms of expected/not expected in a determined time step. On the one hand, this strategy gives a performance-oriented view over the spectrum that maps the transaction to an useful information about the agent and, at the same time, solving the time dependence problem. On the other hand, this solution mainly rely on the precision of a mechanism in detecting an anomalous behaviour in an agent at a given time. This discussion motivates another research question discussed further on in this section. Note that detecting an unexpected behaviour does not necessarily mean that one has identified the agent that is causing the system failure to occur.

In respect to the second dependence, we have proposed a simple yet elegant solution by executing the monitored MAS several time in order to catch multiple instances of agents' behaviour. This solution also addresses the reliance in which the pair (A, e) has on the environment's settings. Furthermore, we have observed that agents are constantly monitored over time but do not consistently fail and, consequently, the collected performance

spectra have low rate of appropriate information to diagnose the faulty agent. So we have proposed an optimisation named MAS-Filter, which increases diagnostic accuracy by excluding low-entropy rows in the spectrum. This is achieved by excluding non-useful time frames.

Looking at the ESFL-MAS algorithm, one can see the intrinsic correlation between the diagnostic report and the similarity coefficient. This leads us to the second research question:

2. *Which similarity coefficients have the best performance given the particular features of agent-based applications?*

SFL assume that closely correlated components are more likely to be relevant to an observed failure and thus it compares transactions over multiple runs further computing suspiciousness values of agents to indicate which of these is the most likely to be the faulty one. This computation is done by the similarity coefficient that sets the diagnostic quality of ESFL-MAS. However, literature has shown that there is no standard similarity coefficient that yields the best result for SFL [Yoo et al., 2014, Hofer et al., 2015, Le et al., 2013]. Aimed to answer this research question, the set of coefficients that excels for the specific context of MASs has been established by empirical evaluation.

Our study investigated an exhaustive list of 42 similarity coefficients. As some of them behave very similarly to others, we used hierarchical clustering together with bootstrapping analysis. Coefficients within the same cluster present low inter-cluster distances and high intra-cluster distances.

Based on the obtained clusters, we have draw two conclusions: (1) from the mathematical formulae of coefficients, there is a logical causal relationship between accuracy and the dichotomy matrix: the more significance a coefficient assigns to anomalous behaviour for which a system error has been detected (represented by c_{11}), the better the diagnostic quality; (2) the test suite has reduced capacity of representing cascading effects of undesired behaviour.

The empirical assessment has demonstrated that for ESFL-MAS the Accuracy, Coverage, Jaccard, Laplace, Least Contradiction, Ochiai, Rogers and Tanimoto, Simple-Matching, Sorensen-Dice, and Support outperform the remainder coefficients across the entire quantity and quality data space (yielding 96.26% diagnostic accuracy) in the specific conditions of our test suite.

The third research question was:

3. *What is the minimal threshold for both the amount of available information and the precision of error detection in which SFL for MASs sustains good performance?*

This research question addresses two independent issues and we have designed experimental evaluations for both of them.

In practice, testers either work under short-time constraints or they do not want to extensively assess the system so there is a constraint in the amount of measurements one can collect from MAS executions. Thus, to represent this in our experiments the number of available passed (N_p) and failed time steps (N_f) were varied and, at the same time, the ESFL-MAS diagnostic quality was monitored. Since the ratio of failed and passed time steps is very small (nearly 0.02), N_p and N_f are varied from 0.001% to 100% of the total number of passed and failed time steps. Results show that adding failed time steps improves the diagnostic quality for most cases and the minimal threshold varies between 200 and 2000 failed time steps. With regard to the number of correct time steps N_p, results show that for the set of similarity coefficients with the best performance, they have marginal influence.

Error detection is the phase that precedes the diagnosis and as such it has great impact on the ESFL-MAS' diagnostic quality. To evaluate how precision in detecting errors influences the diagnostic quality, we varied precision using two methods: (1) excluding time steps that activate the faulty agent, but for which no system error has been detected decreasing c_{10}, and increasing the precision of error detector; and (2) excluding time steps that activate the faulty agent and for each a system error has been detected decreasing c_{11}, and decreasing the precision of error detector. Therefore, we conclude that, on average for all cases, a detection precision greater than 40% have marginal contribution to a better fault diagnosis; nevertheless, for the best group of coefficients ESFL-MAS can achieve high accuracy even for low error detection precision being the borderline precision greater than 10%.

Finally, the last research question has led us to research further on the MAS nature of the fault diagnosis:

4. *Can an SFL-based approach maintain high performance while searching for a faulty agent in various types of organisation and different coordination levels?*

Experimenting ESFL-MAS using only one type of MAS is not enough to show its real potential. In this manner, we have created our experimental setup that has multiple MAS instance besides to providing the necessary information for the aforementioned studies.

The resulting clusters for the NCNO, NCO, CNO, and CO types of MASs created to gather similar coefficients had different number of groups (and their elements) for all cases. There are 7 groups for NCNO and NCO whereas 5 are the groups for CNO and CO. This happens because differences in NCNO and NCO cases are more diverse than in CO and CNO case. An interesting result was that the best group for all types of MASs had the same elements. They were aforementioned while answering research question number one. This initially demonstrated that ESFL-MAS performance was stable regardless the organization or cooperation among agents. Furthermore, other experiment had shown that the number of correct time steps N_p can positively influence the accuracy of ESFL-MAS when agents cooperate as teammates in an organised manner; that is, more examples of nominal behaviour helps the algorithm to effectively pinpoint the faulty agent.

As far as fault detection and diagnosis for MASs are concerned, we have also made an initial contribution towards a sound experimental setup. The test suite was based on the Pickup and Delivery Problem, which is a well-known real-world representative problem. As entities operate upon highly dynamic environments and decisions have to be made under

uncertainty and incomplete knowledge, this problem is most suitable to test agent-based applications. The implementation was based on the MAPC AgentSpeak version of the Goldminers scenario. We improved the number of variants regarding the organisation and coordination of agents; and injected both hand-seeded and mutation-based fault to mimic different kinds of failures. This experimental setup is sufficiently solid and sound to be used as an initial benchmark to fault diagnosis in MASs.

After answering the research questions, it is paramount to deliberate whether or not the thesis' hypothesis hold given the obtained results. First and foremost, let us recall our hypothesis.

Spectrum-based fault localisation algorithms can be adapted to better cope with both the time persistence and the autonomy of multi-agent systems.

Throughout this thesis we have proposed the ESFL-MAS, a model-less diagnosis technique to pinpoint behavioural faults in MASs, and assessed it by testing its stability while varying each of its dependencies. Therefore, the hypothesis is proven within the boundaries of our experimental platform (i.e, closed and homogeneous MASs).

5.2 Further Developments

The spectrum-based technique devised in this research allowed to diagnose unwanted behaviour of a faulty agent in close and homogeneous MAS to mainly support testing in the system level. ESFL-MAS has achieved high diagnostic performance given the boundaries of our experimental setup, so demonstrating the successful adaptation of SFL algorithm to agent-based applications. Even though further developments might be suggested to enhance generalisation and expand the SFL's scope of support within MAS arena.

Investigate generic error detectors. In our empirical assessment, we had emulated an error detection mechanism by defining performance thresholds based on the correct version of each MAS. However, as previously discussed, this is a very domain dependent approach, which goes in the opposite direction of generalisation when applying ESFL-MAS. Aiming at total automation and generalisation of the detection-diagnose process, there is a need to investigate error detectors capable of replacing *a priori* information for a more data-driven orientation. Such anomaly detection approaches have been proposed to MASs (see, e.g. [Chandola et al., 2009, Khalastchi et al., 2015]) and would allow ESFL-MAS to be more easily deployed in other domains.

Improve experimental setup. Although the used experimental setup is an instance of a PDP problem with a validated implementation in AgentSpeak, we are aware of its limitation regarding the complexity of agent coordination and environmental dynamism. The first step towards a better setup is to include the two newest scenarios: Agents on Mars[1] and Agents in the City[2]. Both scenarios would be advantageous as ESFL-MAS (and other diagnosis techniques) could be tested in a heterogeneous MAS with agents acting upon dynamic environments. A much needed modification we would like to increase the size of both scenarios so they could become large-scaled experimental setups. However, this means different instances of the server should be running in various computers, thus increasing the complexity of this implementation. Moreover, there are simulated applications that could supply a broader range of MASs besides the PDPs. Some examples are efforts on artificial traffic control system [Rossetti et al., 2008], real-time traffic control[3] [Vilarinho et al., 2015], and schedule crane operations using agents [Heshmati et al., 2016]. Another limitation we must address is the reduced capacity that the current experimental setup has of representing cascading effects of unwanted behaviours. The best way to emulate this kind of fault is to use mocking agents as suggested by Coelho et al. [2007, 2006]. It induces failures in other agents by, for instance, providing wrong information. This strategy is perhaps the most effective on stimulating masked faults and malicious behaviours.

Apply ESFL-MAS to realistic deployments. In the introduction of this research, we support the idea that more reliable MASs would increase their acceptance by the industrial community. However, it is naive to expect that practitioners will trust assessments and (even good) results based on so-called toy problems. Proposed approaches should prove their value while dealing with real threats to surpass the industry's scepticism. One may notice similarities between this further development and the later. and question the reason to separate their discussions. We consider both distinct efforts. The latter demands a rather scientific rationale, envisioning to determine boundaries of performance for fault diagnosis techniques. This endeavour requires the implementation of more application-specific modules such as: code instrumentation, behaviour and message monitoring, and so forth. Centered on the application of the ESFL-MAS to realistic MASs deployments, there are several of such systems developed within our research centre and collaboration network, such as: IntellWheels [Braga et al., 2011], MAS to support Power Distribution System Operation [Issicaba et al., 2012], MAS-T^{2}er Lab [Rossetti et al., 2007], MASDIMA[4] [Castro et al., 2014], GRACE[5] [Rodrigues et al., 2013].

Explore ESFL-MAS in the component level of agents. We have mainly focused on studying the fundamental limitations of SFL to diagnose faults at the agent level. Although such an approach is valuable to understand how a faulty agent may impact the

[1] In Agents on Mars scenario, the MAS aims to secure as much space on Mars as possible using a team of cooperative agents; there are five roles with different properties and abilities.

[2] In Agents in the City scenario, the MAS' goal is collect as much money as possible by completing jobs across a virtual city; again there are various roles, but in this scenario agents act upon a dynamic scenario.

[3] This work would demand for more implementation because it deals with a traffic simulator (from which we can have environmental measures) and TraSMAPI [Timóteo et al., 2012] (that supply information about the agent's state).

[4] `http://masdima.com`.

[5] `http://grace-project.org`.

overall performance, applying the ESFL-MAS using block-hit spectra to localise fault inside agent modules would give us a whole new perspective about SFL's limitations/trends. This research line could potentially aid the implementation of more generic debugging tools to MAS. Another application would be a complementary mechanism to diagnose faults in agents' plans, similarly to the work of Micalizio [2013].

Incorporate interaction information into ESFL-MAS. Interaction is a huge part in the MAS area as agents are spatially distributed entities in an environment. In a highly interactive agent-based system, cascading faults might cause low performance or even overall failure due to malfunctioning in communication, ontology flaws, and so forth. Encompassing this information would improve the diagnostic accuracy of the detection-diagnosis process; for example, Kalech [2012] proposes a matrix-based approach to diagnose coordination faults. In the case of spectrum-based approaches, the interaction among agents could be used as a weak and dynamic dependence model that would be a factor to consider in the diagnostic reasoning. Of course, some overhead regarding the message monitoring is going to be added, but further research on compression techniques could emerge to optimisation purposes. Specifically in MASs, it is almost mandatory that a fault diagnosis tool supports FIPA-ACL protocols, which are the standards for agents communication; however, other communications infrastructures (such as SACI [Hubner and Sichman, 2003]) should be taken into account.

Use spectrum-based reasoning to support multiple faults. ESFL-MAS yields good diagnostic accuracy, however it is inherently limited since no reasoning to localise the fault is applied. Many of spectrum-based diagnosis limitations have been handle by the software engineering community, such as: (1) it can identify multiple faults [Abreu et al., 2009], (2) it can aggregate faults scattered across the code [Wong et al., 2012], (3) it can quantify confidence in the diagnosis [de Kleer, 2009, Cardoso and Abreu, 2013], and (4) it can deal with non-functional errors [Cardoso and Abreu, 2014]. These improvement could potentially aid localising more complex faults in agents' behaviour. Machine learning and case-based reasoning [Aamodt and Plaza, 1994, Kolodner, 1992] can be used to analyse the correlation of transaction within a time window and across similar test cases, respectively. Hence, together with a more generic anomaly detection scheme, new challenges can arise from the application of such approaches in the MAS context.

5.3 Research Trends and Challenges

Ensuring nominal MAS performance has been an evergrowing and stimulating field of study that has continuously raised challenges and research issues over the past decade. Relevant examples in the literature, as discussed in Chapter 2, illustrate the potential and need of further research in the area. In the following paragraphs, we discuss in the following open research issues and trends in techniques to increase reliability of MASs over its whole life cycle, for the next decades.

Exhaustive benchmarking. A wide range of systems and techniques has been proposed over the past few years. However, across the literature there are recognised problems with comparing and even experimenting the methods developed. This is due mainly to the use of different software implementations, different applications, but also because of the lack of solid metrics in many studies. As a result, many of the published approaches present a profound lack of experimental evidences that could convince designers to use them, and comparison between them is not a priority for those which have been assessed. All of this calls for a comprehensive, extensive benchmark involving (1) toy problems that evaluate conceptual boundaries of techniques, (2) complex applications of major domains in which MASs have been deployed, and (3) metrics that may be used for a fair comparison among approaches. Of course building such a benchmark is not an easy task, but there are already efforts in such a direction. For instance, the Multi-Agent Programming Contest (MAPC) is an annual competition that, since 2005, aims to provide suitable test suite offering key problems for the community to test with agent-oriented programming approaches [Behrens et al., 2010, Ahlbrecht et al., 2013]. Dealing with larger scale problems, Maffioletti et al. [2013] focus on the benchmarking of coordination in urban search and rescue scenarios. Although the benchmark does not seem at first glance a crucial component of reliable MAS, the sound, progressive, and solid implementation and deployment of such schemes will be constrained by the challenge of building exhaustive benchmark.

Combining techniques. The idea of combining techniques goes back to the beginning of reliable MASs. Since then, many authors have suggested the superiority of combining mechanisms over the use of especialised ones. For instance, the replication framework DARX [Guessoum et al., 2010] has been recently combined with the exception handling framework SaGE [Dony et al., 2011] increasing fault tolerance coverage of the resulting framework. Similarly, PDT [Padgham et al., 2013] associates testing schemes from several levels and as such expands their usefulness. Despite this popularity, the combination of techniques has not been extensively discussed across the development cycle. Winikoff [2010] discusses the combination of testing and proving arguing that the strength of testing is to be able to deal with concrete implementation whereas proving can certify abstract models. We go beyond and include fault tolerance techniques as a crucial stage after the MAS deployment as it can avoid failure situations during the operation phase. Only with these three pieces working in synergy, MAS reliability is covered thoroughly during its complete life cycle. The integration of these pieces is still an open issue and a rich soil for further research.

Automated validation. eCAT [Nguyen et al., 2012], PDT [Padgham et al., 2013], and SEAUnit [Çakirlar et al., 2009] frameworks have applied automation up to some extent, but the majority of simulation-based design validation efforts relies on designers to provide useful information and/or assess results of MASs. This dependence on the designer increases labor, error-proneness, and costs. A fully automated testing and validation tool would be able to monitor agents and interaction patterns, and from them formulate the expected behaviour; it would then run the system under several environmental conditions, compare results with the built behavioural model, and report the success or failure of these automatically generated tests. As we can see by the reviewed literature, there are still open challenges so full automation can be achieved for MAS testing and validation. First, the proposal of techniques that build agent behavioural model without having access to its

internal modules by monitoring perceptions, actions, and messages. Second, mechanisms that are able to detect anomalies in social interaction of agents given only the history of such interactions. Finally, the community should study new methods to measure MAS performance that are neither solely based on the overall utility nor are specific to certain applications.

Prognostic framework. Adaptability and evolution are two specially appealing features of MASs. The former concerns short-term events and can be two distinctive things: the ability of an agent to learn and adapt its behaviour to a given event, or the ability of part of the system to restructure given environmental changes. The latter regards to medium- and long-term situations and have a fundamental difference from adaptation. After an unforeseen event ends, in adaptive agent/MAS, the system returns to its original state as before the event; however, if the system evolves, it can no longer come back to its previous state. A prognostic framework addresses issues, such as: (1) how discriminate anomalous behaviour from adaptation; (2) how to identify emergent behaviour and forecast its consequences in the overall performance; (3) how to guide evolution to achieve quasi-optimal results, and so forth. Fortunately, some interesting research on drift adaptation in time series[6] might provide initial guidelines towards solving the aforementioned issues. As one can see, this is perhaps the most long-term line of research pointed out in this thesis, even though it is closely related to the previous trend.

[6]Readers seeking for extensive review on drift adaptation on time series can consult Gama et al. [2014]

Appendix A

Marginal Research Efforts

This appendix describes the marginal research efforts developed along with the main thesis theme during the Ph.D. studies. First, the project *A Multi-Agent Platform to Support Ubiquitous Transportation Systems* is the previous Ph.D. theme, aimed to supply ubiquitousness through agents in future urban transportation. Second, the project *A Platform for the Design, Simulation and Development of Quadcopter MASs* aims to provide a toolbox for designers studying autonomous vehicles from the bottom level (e.g., sensors and motors) up to higher levels (e.g., decision making processes). Further details will be given for each project in the next sections.

A.1 A Multi-Agent Platform to Support Ubiquitous Transportation Systems

In recent years, the population in major urban areas grew and increased in number of cars, becoming traffic chaotic in these areas. The problem of traffic congestion not only affects the day-to-day life of citizens but also has a great impact on business and economic activities. Fearing the consequences of these issues in mid and long terms, both practitioners and the scientific communities have striven to tackle congestion in such large urban networks.

Much effort has been put on the design and specification of the future transport systems, putting the user in the centre of all concerns and largely oriented to services. Such efforts culminated in the emergence of the concept of Intelligent Transportation Systems, basically relying on a distributed and advanced communication infrastructure favoring interactions in virtually all levels. It suggests a completely new decentralised perspective of processes; however, discussions are still fostered by current ambitions at this new paradigm.

With the emergence the role of electronic devices in people's lives, a new concept, called Ubiquitous Computing, was firstly pointed out by Mark Weiser. It basically relied on the use of computer resources and provision of information and services to people whenever and wherever they were requested and needed. Furthermore, Ubiquitous Transportation Systems are becoming a reality because of the increasing use of computer resources to

supply continuous connectivity to such a domain. Multi-Agent Systems have been proving to be an important ingredient to support ubiquitousness, favouring mobility, information acquisition and sharing, awareness and cooperation. The main reason is their ability to provide autonomous decisions and high-level interactions between entities.

These three concepts, despite being potentially complementary, have not as yet been dealt with on an integrated basis. Indeed, a diverse range of different applications and for varying purposes have already been reported in the literature, coupling the two of them and opening up a challenging and promising novel field of research.

This work proposes the extinction of the barrier between demand and supply in transportation systems, as suggested by researchers in the field. To achieve this ubiquitous computing and other related concepts can be applied to merge the supply and demand, providing support the effective implementation of future transportation systems requirements. Concretely speaking, the work aims to specify, devise and implement a framework featuring the necessary mechanisms underlying the full characteristics of ubiquitous transportation systems, which are presented as a natural perspective of today's intelligent transportation technologies supported by major multi-agent systems features.

This project generated many research outcomes that are listed below:

1. L. S. Passos, R. J. F. Rossetti, J. Gabriel, *An Agent Methodology for Processes, the Environment, and Services.* In Advances in Artificial Transportation Systems and Simulation, pages 37-53, Rosaldo J. F. Rossetti and Ronghui Liu (eds.) Academic Press, Boston. 2015. [Passos et al., 2015b].

2. L. S. Passos, Z. Kokkinogenis, R. J. F. Rossetti, J. Gabriel, *Multi-Resolution Simulation of Taxi Services on Airport Terminal's Curbside.* In Proceedings of the 15th IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC 2013), The Hague, The Netherlands, October. 2013. [Passos et al., 2013a].

3. Z. Kokkinogenis, L. S. Passos, R. J. F. Rossetti, J. Gabriel, *Towards the next-generation traffic simulation tools: a first evaluation.* In Proceedings of the 6th Doctoral Symposium in Informatics Engineering (DSIE 2011), Porto, Portugal, January 27-28. 2011. [Kokkinogenis et al., 2011].

4. L. S. Passos, Z. Kokkinogenis, R. J. F. Rossetti, *Towards the next-generation traffic simulation tools: a first appraisal.* In Proceedings of the 3th Workshop on Intelligent Systems and Applications (WISA 2011), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2011), Chaves, Portugal, June 15-18. 2011. [Passos et al., 2011a].

5. L. S. Passos, R. J. F. Rossetti, L. P. Reis, *Evaluation of taxi services on airport terminal's curbside for picking up passengers.* In Proceedings of the 3th Workshop on Intelligent Systems and Applications (WISA 2011), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2011), Chaves, Portugal, June 15-18. 2011. [Passos et al., 2011b].

6. L. S. Passos, R. J. F. Rossetti, *Traffic Light Control using Reactive Agents*. In Proceedings of the 2nd Workshop on Intelligent Systems and Applications (WISA 2010), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2010), Santiago de Compostela, Spain, June 16-19. 2010. [Passos and Rossetti, 2010a].

7. L. S. Passos, R. J. F. Rossetti, E. Oliveira, *Ambient-Centred Intelligent Traffic Control and Management*. In Proceedings of the 13th IEEE International Conference on Intelligent Transportation Systems (IEEE ITSC 2010), Madeira Island, Portugal, September 19-22. 2010. [Passos et al., 2010].

8. L. S. Passos, *Evaluating Agent-based Methodologies with Focus on Transportation Systems*. In Proceedings of the 5th Doctoral Symposium in Informatics Engineering (DSIE 2010), Porto, Portugal, January 28-29. 2010. [Passos, 2010].

9. L. S. Passos, R. J. F. Rossetti, *Intelligent Transportation Systems: a Ubiquitous Perspective*. In New Trends in Artificial Intelligence collocated in the Proceedings of the 14th Portuguese Conference on Artificial Intelligence (EPIA), Aveiro, Portugal. 2009. [Passos and Rossetti, 2009].

A.2 A Platform for the Development of Quadcopter MASs

Unmanned Aerial Vehicles gained notoriety in the past decade due to their successful endeavours in warfare. Since then, many civil applications appeared aiming to use this technology to substitute the human operators in hazardous situations. Indeed, the research directions in their development nowadays seek to consolidate three interesting features: a higher-level decision-making process, autonomy in taking actions, and coordination of efforts towards a common goal in the case of applications with multiple units. Such vehicles in general (and specially because of the aforementioned features) demand proper simulation tools in order to avoid damage to expensive equipment and misuse of means that could be employed to improve their development life-cycle.

Our research focuses on the coordinated usage of *quadcopters* in both indoor and outdoor environments; an illustrative scenario would be a group of quadcopters helping to map the internal blueprint of a building when it is dangerous for people to go inside. This type of aerial vehicle suits indoor applications due to its manoeuvrability (i.e. it changes direction and altitude faster than aeroplanes), as well as it easily stabilizes compared to other types of Unmanned Aerial Vehicles. On the other hand, quadcopters usually struggles with their flying time as designers have to optimize flying weight versus battery life. From the simulation perspective, all the aforementioned aspects must be present in the virtual domain so quadcopters' planning, learning, and other features might be put under test accounting for realistic conditions.

A question emerges upon these considerations: which approach does fulfil the requirements for a collaborative quadcopters simulation tool for real indoor and outdoor applications? We believe that the answer may lie within the *symbiotic simulation* paradigm. Such per-

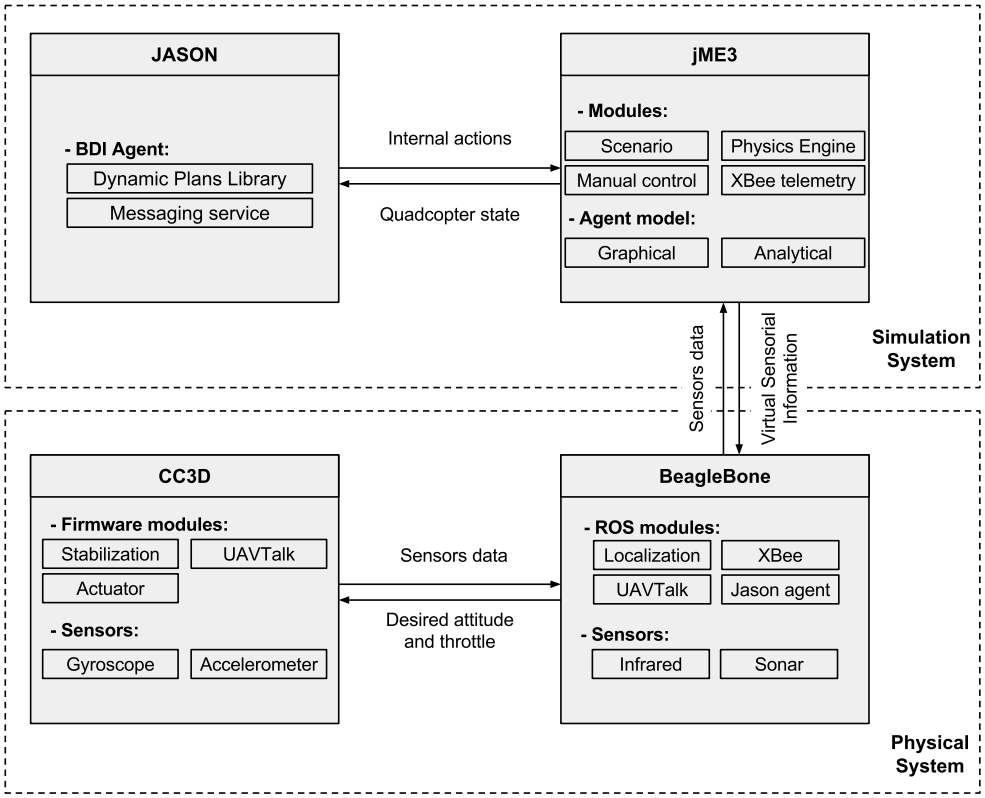


Figure A.1: The proposed framework for Quadcopter MASs.

spective considers a close association between a virtual (simulation) system and a physical system as well as the mutual benefits can emerge from this synergy. We explore the various classes of symbiotic simulation systems with the purposes of (1) controlling the real quadcopter, (2) supporting an external cognitive decision maker, (3) training the quadcopter in different environments, (4) validating the virtual model and strategies, and (5) detecting anomalies in both virtual and physical systems.

This work proposes an architecture to support symbiotic simulation of quadcopters aiming to assess techniques from low-level control to coordination and cooperation strategies (Figure A.1). That is, a platform for the design, simulation, and development of quadcopter Multi-Agent Systems. We study the requirements of our quadcopter context from the perspective of symbiotic simulation and thus illustrate some scenarios to base the proposed architecture. A main contribution of this work is to describe all components in our abstract architecture and how the chosen technologies play their roles in each component; also we show some preliminary results that demonstrate all distinct technologies are able to interact in a cohesive manner.

This project generated some research outcomes that are listed below:

1. R. Veloso, G. Oliveira, Z. Kokkinogenis, L. S. Passos, R. J. F. Rossetti, J. Gabriel, *A Symbiotic Simulation Platform for Agent-based Quadcopters*. In Proceedings of the 9th Iberian Conference on Information Systems and Technologies (CISTI 2014),

Barcelona, Spain, June 18-21. 2014. [Veloso et al., 2014b].

2. R. Veloso, Z. Kokkinogenis, L. S. Passos, G. Oliveira, R. J. F. Rossetti, J. Gabriel, *A Platform for the Design, Simulation and Development of Quadcopter Multi-Agent Systems*. In: Proceedings of the 9th Iberian Conference on Information Systems and Technologies (CISTI 2014), Barcelona, Spain, June 18-21. 2014. [Veloso et al., 2014a].

References

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59. (Cited on page 87.)

[Abreu and Van Gemund, 2010] Abreu, R. and Van Gemund, A. J. (2010). Diagnosing multiple intermittent failures using maximum likelihood estimation. *Artificial Intelligence*, 174(18):1481–1497. (Cited on page 55.)

[Abreu et al., 2009] Abreu, R., Zoeteweij, P., Golsteijn, R., and van Gemund, A. J. (2009). A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software (JSS)*, 82(11):1780 – 1792. SI: TAIC PART 2007 and MUTATION 2007. (Cited on pages 5, 39, 49, 78, 82 and 87.)

[Agatz et al., 2012] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295 – 303. (Cited on pages 60 and 63.)

[Agogino and Tumer, 2012] Agogino, A. and Tumer, K. (2012). A multiagent approach to managing air traffic flow. *Autonomous Agents and Multi-Agent Systems*, 24(1):1–25. (Cited on page 1.)

[Ahlbrecht et al., 2013] Ahlbrecht, T., Dix, J., Köster, M., and Schlesinger, F. (2013). Multi-agent programming contest 2013. In Cossentino, M., El Fallah Seghrouchni, A., and Winikoff, M., editors, *Engineering Multi-Agent Systems*, volume 8245 of *Lecture Notes in Computer Science*, pages 292–318. Springer-Verlag. (Cited on page 88.)

[Almeida et al., 2008] Almeida, A. d. L., Aknine, S., and Briot, J.-P. (2008). Dynamic resource allocation heuristics for providing fault tolerance in multi-agent systems. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 66–70, New York, NY, USA. ACM. (Cited on page 27.)

[Avižienis et al., 2004] Avižienis, A., Laprie, J.-C., and Randell, B. (2004). Dependability and its threats: A taxonomy. In Jacquart, R., editor, *Building the Information Society*, volume 156 of *IFIP International Federation for Information Processing*, pages 91–120. Springer-Verlag. (Cited on pages xv, 2, 13, 14, 15 and 48.)

[Bazzan and Klügl, 2013] Bazzan, A. L. C. and Klügl, F. (2013). A review on agent-based technology for traffic and transportation. *The Knowledge Engineering Review*, FirstView:1–29. (Cited on page 1.)

[Behrens et al., 2010] Behrens, T., Dastani, M., Dix, J., Köster, M., and Novák, P. (2010). The multi-agent programming contest from 2005-2010. *Annals of Mathematics and Artificial Intelligence*, 59(3-4):277–311. (Cited on page 88.)

[Bernon et al., 2007] Bernon, C., Gleizes, M.-P., and Picard, G. (2007). Enhancing self-organising emergent systems design with simulation. In O'Hare, G. M., Ricci, A., O'Grady, M. J., and Dikenelli, O., editors, *Engineering Societies in the Agents World VII*, volume 4457 of *Lecture Notes in Computer Science*, pages 284–299. Springer-Verlag. (Cited on pages 21, 30, 32, 35 and 37.)

[Bhattacharya and Waymire, 1990] Bhattacharya, R. N. and Waymire, E. C. (1990). *Stochastic Processes with Applications*. John Wiley & Sons. (Cited on page 45.)

[Bond and Gasser, 1988] Bond, A. H. and Gasser, L. G. (1988). An analysis of problems and research in dai. In Bond, A. H. and Gasser, L. G., editors, *Readings in Distributed Artificial Intelligence*, chapter 1, pages 3–35. Morgan Kaufmann Publishers Inc., San Mateo, CA. (Cited on page 1.)

[Bordini et al., 2004] Bordini, R. H., Fisher, M., Visser, W., and Wooldridge, M. (2004). Verifiable multi-agent programs. In Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Programming Multi-Agent Systems*, volume 3067 of *Lecture Notes in Computer Science*, pages 72–89. Springer-Verlag. (Cited on page 12.)

[Botía et al., 2006] Botía, J. A., Gómez-Sanz, J. J., and Pavón, J. (2006). Intelligent data analysis for the verification of multi-agent systems interactions. In Corchado, E., Yin, H., Botti, V., and Fyfe, C., editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2006*, volume 4224 of *Lecture Notes in Computer Science*, pages 1207–1214. Springer-Verlag. (Cited on page 19.)

[Braga et al., 2011] Braga, R. A. M., Petry, M., Reis, L. P., and Moreira, A. P. (2011). Intellwheels: Modular development platform for intelligent wheelchairs. *Journal of Rehabilitation Research & Development*, 48(9):1061 – 1076. (Cited on page 86.)

[Campos et al., 2013] Campos, J., Abreu, R., Fraser, G., and d'Amorim, M. (2013). Entropy-based test generation for improved fault localization. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 257–267. (Cited on page 49.)

[Cao et al., 2009] Cao, L., Gorodetsky, V., and Mitkas, P. A. (2009). Agent mining: The synergy of agents and data mining. *IEEE Intelligent Systems*, 24(3):64–72. (Cited on page 1.)

[Cardoso and Abreu, 2013] Cardoso, N. and Abreu, R. (2013). A kernel density estimate-based approach to component goodness modeling. In *AAAI Conference on Artificial Intelligence*. (Cited on page 87.)

[Cardoso and Abreu, 2014] Cardoso, N. and Abreu, R. (2014). Enhancing reasoning approaches to diagnose functional and non-functional errors. In *25th International Workshop on Principles of Diagnosis*. (Cited on page 87.)

[Carrera et al., 2012] Carrera, Á., Iglesias, C. A., García-Algarra, J., and Kolařík, D. (2012). A real-life application of multi-agent systems for fault diagnosis in the provision of an internet business service. *Journal of Network and Computer Applications*. (Cited on page 1.)

[Casanova et al., 2013] Casanova, P., Garlan, D., Schmerl, B., and Abreu, R. (2013). Diagnosing architectural run-time failures. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pages 103–112, Piscataway, NJ, USA. IEEE Press. (Cited on page 49.)

[Castro et al., 2014] Castro, A. J. M., Rocha, A. P., and Oliveira, E. (2014). *A New Approach for Disruption Management in Airline Operations Control*, volume 562 of *Studies in Computational Intelligence*. Springer-Verlag. (Cited on page 86.)

[Çakirlar et al., 2009] Çakirlar, I., Ekinci, E., and Dikenelli, O. (2009). Exception handling in goal-oriented multi-agent systems. In Artikis, A., Picard, G., and Vercouter, L., editors, *Engineering Societies in the Agents World IX*, volume 5485 of *Lecture Notes in Computer Science*, pages 121–136. Springer-Verlag. (Cited on pages 19, 30, 32, 35, 37 and 88.)

[Çakirlar et al., 2015] Çakirlar, I., Gürcan, O., Dikenelli, O., and Bora, c. (2015). Ratkit: Repeatable automated testing toolkit for agent-based modeling and simulation. In Grimaldo, F. and Norling, E., editors, *Multi-Agent-Based Simulation XV*, volume 9002 of *Lecture Notes in Computer Science*, pages 17–27. Springer-Verlag. (Cited on pages 7, 22, 30, 32, 35, 36 and 37.)

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Survey*, 41(3):15:1–15:58. (Cited on pages 51 and 85.)

[Charette, 2005] Charette, R. (2005). Why software fails [software failure]. *Spectrum, IEEE*, 42(9):42–49. (Cited on pages 2 and 11.)

[Chia et al., 1998] Chia, M. H., Neiman, D. E., and Lesser, V. R. (1998). Poaching and distraction in asynchronous agent activities. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 88 –95. (Cited on pages 4, 23, 30, 32, 35 and 37.)

[Coelho et al., 2007] Coelho, R., Cirilo, E., Kulesza, U., von Staa, A., Rashid, A., and Lucena, C. (2007). Jat: A test automation framework for multi-agent systems. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 425–434. (Cited on pages 4, 18, 30, 32, 35, 37 and 86.)

[Coelho et al., 2006] Coelho, R., Kulesza, U., von Staa, A., and Lucena, C. (2006). Unit testing in multi-agent systems using mock agents and aspects. In *Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems*, SELMAS '06, pages 83–90, New York, NY, USA. ACM. (Cited on pages 18 and 86.)

[Console and Torasso, 1991] Console, L. and Torasso, P. (1991). A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141. (Cited on page 5.)

[Cossentino, 2005] Cossentino, M. (2005). From requirements to code with passi methodology. In Henderson-Sellers, B. and Giorgini, P., editors, *Agent-Oriented Methodologies*, pages 79–106. (Cited on page 21.)

[Cossentino et al., 2008] Cossentino, M., Fortino, G., Garro, A., Mascillaro, S., and Russo, W. (2008). Passim: a simulation-based process for the development of multi-agent systems. *International Journal of Agent-Oriented Software Engineering*, 2(2):132–170. (Cited on pages 7, 21, 30, 31, 32, 35 and 37.)

[Cossentino et al., 2010] Cossentino, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010). Aspecs: an agent-oriented software process for engineering complex systems. *International Journal of Autonomous Agents and Multi-Agent Systems (IJAAMAS)*, 20(2):260–304. (Cited on page 13.)

[Cruz-Correia et al., 2005] Cruz-Correia, R., Vieira-Marques, P., Costa, P., Ferreira, A., Oliveira-Palhares, E., Araújo, F., and Costa-Pereira, A. (2005). Integration of hospital data using agent technologies - a case study. *AI Commun.*, 18(3):191–200. (Cited on page 1.)

[Dastani et al., 2007] Dastani, M., Dix, J., and Novák, P. (2007). The second contest on multi-agent systems based on computational logic. In Inoue, K., Satoh, K., and Toni, F., editors, *Computational Logic in Multi-Agent Systems*, volume 4371 of *Lecture Notes in Computer Science*, pages 266–283. Springer-Verlag. (Cited on page 60.)

[de Kleer, 2009] de Kleer, J. (2009). Diagnosing multiple persistent and intermittent faults. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, pages 733–738, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (Cited on page 87.)

[de Kleer et al., 1992] de Kleer, J., Mackworth, A. K., and Reiter, R. (1992). Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2):197 – 222. (Cited on page 5.)

[de Kleer and Williams, 1987] de Kleer, J. and Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97 – 130. (Cited on pages 5 and 51.)

[De Wolf et al., 2006] De Wolf, T., Holvoet, T., and Samaey, G. (2006). Development of self-organising emergent applications with simulation-based numerical analysis. In Brueckner, S. A., Di Marzo Serugendo, G., Hales, D., and Zambonelli, F., editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 138–152. Springer-Verlag. (Cited on pages 4, 21, 30, 31, 32, 35 and 37.)

[Dellarocas and Klein, 2000] Dellarocas, C. and Klein, M. (2000). An experimental evaluation of domain-independent fault handling services in open multi-agent systems. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 95 –102. (Cited on pages 4 and 26.)

[Demazeau et al., 2015] Demazeau, Y., Decker, K. S., Pérez, J. B., and de la Prieta, F., editors (2015). *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability: The PAAMS Collection*. Springer-Verlag, Salamanca, Spain. (Cited on page 1.)

[Dennis et al., 2014] Dennis, L. A., Fisher, M., Lincoln, N. K., Lisitsa, A., and Veres, S. M. (2014). Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering*, pages 1–55. (Cited on page 4.)

[Dennis et al., 2015] Dennis, L. A., Fisher, M., and Webster, M. (2015). Two-stage agent program verification. *Journal of Logic and Computation*. (Cited on page 4.)

[Dennis et al., 2012] Dennis, L. A., Fisher, M., Webster, M. P., and Bordini, R. H. (2012). Model checking agent programming languages. *Automated Software Engg.*, 19(1):5–63. (Cited on page 12.)

[Dignum et al., 2002] Dignum, V., Meyer, J.-J., Weigand, H., and Dignum, F. (2002). An organizational-oriented model for agent societies. In *Proc. Int. Workshop on Regulated Agent-Based Social Systems: Theories and Applications (RASTA'02), at AAMAS, Bologna, Italy*. (Cited on page 1.)

[Dikenelli et al., 2006] Dikenelli, O., Erdur, R. C., Kardas, G., Gümüs, O., Seylan, I., Gürcan, O., Tiryaki, A., and Ekinci, E. E. (2006). Developing multi agent systems on semantic web environment using seagent platform. In Dikenelli, O., Gleizes, M.-P., and Ricci, A., editors, *Engineering Societies in the Agents World VI*, volume 3963 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag. (Cited on page 19.)

[Dony et al., 2011] Dony, C., Kchir, S., Tibermacine, C., Urtado, C., Vauttier, S., Ductor, S., and Guessoum, Z. (2011). Combining exception handling and replication for improving the reliability of agent software. Tech report. (Cited on pages 27 and 88.)

[Dony et al., 2008] Dony, C., Tibermacine, C., Urtado, C., and Vauttier, S. (2008). Specification of an exception handling system for a replicated agent environment. In *Proceedings of the 4th international workshop on Exception handling*, WEH '08, pages 24–31, New York, NY, USA. ACM. (Cited on page 27.)

[Ekinci et al., 2009] Ekinci, E. E., Tiryaki, A. M., Çetin, O., and Dikenelli, O. (2009). Goal-oriented agent testing revisited. In Luck, M. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 173–186. Springer-Verlag. (Cited on page 19.)

[Faci et al., 2006] Faci, N., Guessoum, Z., and Marin, O. (2006). Dimax: a fault-tolerant multi-agent platform. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, SELMAS '06, pages 13–20, New York, NY, USA. ACM. (Cited on pages 15 and 27.)

[Fedoruk and Deters, 2002] Fedoruk, A. and Deters, R. (2002). Improving fault-tolerance by replicating agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, AAMAS '02, pages 737–744, New York, NY, USA. ACM. (Cited on page 27.)

[Fedoruk and Deters, 2003] Fedoruk, A. and Deters, R. (2003). Improving fault-tolerance in mas with dynamic proxy replicate groups. In *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, IAT '03, pages 364—-, Washington, DC, USA. IEEE Computer Society. (Cited on pages 27, 30, 32, 35 and 37.)

[Ferber, 1999] Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison Wesley. (Cited on page 47.)

[Fischer, 1999] Fischer, K. (1999). Holonic multiagent systems - theory and applications. In Barahona, P. and Alferes, J. J., editors, *Progress in Artificial Intelligence*, volume 1695 of *Lecture Notes in Computer Science*, pages 34–48. Springer Berlin Heidelberg. (Cited on page 13.)

[Fischer et al., 1996] Fischer, K., Müller, J. P., and Pischel, M. (1996). Cooperative transportation scheduling: An application domain for dai. *Applied Artificial Intelligence*, 10(1):1–34. (Cited on page 60.)

[Fisher et al., 2007] Fisher, M., Bordini, R. H., Hirsch, B., and Torroni, P. (2007). Computational logics and agents: A road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91. (Cited on page 39.)

[Fisher et al., 2013] Fisher, M., Dennis, L., and Webster, M. (2013). Verifying autonomous systems. *Commun. ACM*, 56(9):84–93. (Cited on page 4.)

[Fortino and North, 2013] Fortino, G. and North, M. J. (2013). Simulation-based development and validation of multi-agent systems: Aose and abms approaches. *J Simulation*, 7(3):137–143. (Cited on pages 12 and 21.)

[Fortino et al., 2014] Fortino, G., Rango, F., and Russo, W. (2014). Eldameth design process. In Cossentino, M., Hilaire, V., Molesini, A., and Seidita, V., editors, *Handbook on Agent-Oriented Design Processes*, pages 115–139. Springer-Verlag. (Cited on pages 7, 22, 30, 31, 32, 35 and 37.)

[Fortino and Russo, 2012] Fortino, G. and Russo, W. (2012). Eldameth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems. *Information and Software Technology*, 54(6):608 – 624. Special Section: Engineering Complex Software Systems through Multi-Agent Systems and Simulation Special Section: Engineering Complex Software Systems through Multi-Agent Systems and Simulation. (Cited on page 22.)

[Gama et al., 2014] Gama, J. a., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37. (Cited on page 89.)

[Gardelli et al., 2008] Gardelli, L., Viroli, M., Casadei, M., and Omicini, A. (2008). Designing self-organising environments with agents and artefacts: a simulation-driven approach. *Int. J. Agent-Oriented Softw. Eng.*, 2(2):171–195. (Cited on pages 21, 30, 32, 35 and 37.)

[Gardelli et al., 2006] Gardelli, L., Viroli, M., and Omicini, A. (2006). On the role of simulations in engineering self-organising mas: The case of an intrusion detection system in tucson. In Brueckner, S. A., Di Marzo Serugendo, G., Hales, D., and Zambonelli, F., editors, *Engineering Self-Organising Systems*, volume 3910 of *Lecture Notes in Computer Science*, pages 153–166. Springer-Verlag. (Cited on page 21.)

[Gasser, 1987] Gasser, L. (1987). Distribution and coordination of tasks among intelligent agents. In *First Scandinavian Conference on Artificial Intelligence*, Tromsø, Normay. (Cited on pages 1 and 11.)

[Gómez-Sanz et al., 2009] Gómez-Sanz, J. J., Botía, J., Serrano, E., and Pavón, J. (2009). Testing and debugging of mas interactions with ingenias. In Luck, M. and Gómez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 199–212. Springer-Verlag. (Cited on pages 19, 29, 30, 31, 32, 35 and 37.)

[Gómez-Sanz et al., 2008] Gómez-Sanz, J. J., Fuentes, R., Pavón, J., and García-Magariño, I. (2008). Ingenias development kit: A visual multi-agent system development environment (demo paper). In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers*, AAMAS '08, pages 1675–1676, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on page 19.)

[Gonzalez-Sanchez et al., 2011] Gonzalez-Sanchez, A., Abreu, R., Gross, H.-G., and van Gemund, A. J. C. (2011). Spectrum-based sequential diagnosis. In *AAAI Conference on Artificial Intelligence*. (Cited on page 49.)

[Grunow et al., 2005] Grunow, M., Günther, H.-O., and Lehmann, M. (2005). Dispatching multi-load agvs in highly automated seaport container terminals. In Günther, H.-O. and Kim, K., editors, *Container Terminals and Automated Transport Systems*, pages 231–255. Springer-Verlag. (Cited on pages 60 and 63.)

[Guessoum and Briot, 1999] Guessoum, Z. and Briot, J.-P. (1999). From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76. (Cited on page 27.)

[Guessoum et al., 2010] Guessoum, Z., Briot, J.-P., Faci, N., and Marin, O. (2010). Towards reliable multi-agent systems: An adaptive replication mechanism. *Multiagent Grid Syst.*, 6(1):1–24. (Cited on pages 27, 29, 30, 32, 35, 36, 37 and 88.)

[Guessoum et al., 2003] Guessoum, Z., Briot, J.-P., Marin, O., Hamel, A., and Sens, P. (2003). Dynamic and adaptive replication for large-scale reliable multi-agent systems. In Garcia, A., Lucena, C., Zambonelli, F., Omicini, A., and Castro, J., editors, *Software engineering for large-scale multi-agent systems*, pages 182–198. Springer-Verlag, Berlin, Heidelberg. (Cited on page 27.)

[Gupta et al., 2014] Gupta, S., Abreu, R., de Kleer, J., and van Gemund, A. (2014). Automatic systems diagnosis without behavioral models. In *Aerospace Conference, 2014 IEEE*, pages 1–8. (Cited on page 5.)

[Gürcan et al., 2011] Gürcan, O., Dikenelli, O., and Bernon, C. (2011). Towards a generic testing framework for agent-based simulation models. In *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, pages 635–642. (Cited on page 22.)

[Gürcan et al., 2013] Gürcan, Ö., Dikenelli, O., and Bernon, C. (2013). A generic testing framework for abms models. *J. Simulation*, 7(3):183–201. (Cited on page 22.)

[Hägg, 1997] Hägg, S. (1997). A sentinel approach to fault handling in multi-agent systems. In Zhang, C. and Lukose, D., editors, *Multi-Agent Systems Methodologies and Applications*, volume 1286 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag. (Cited on pages 25, 30, 32, 35 and 37.)

[Hailpern and Santhanam, 2002] Hailpern, B. and Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, 41(1):4–12. (Cited on page 3.)

[Harrold et al., 2000] Harrold, M. J., Rothermel, G., Sayre, K., Wu, R., and Yi, L. (2000). An empirical investigation of the relationship between spectra differences and regression faults. *Software Testing, Verification and Reliability*, 10(3):171–194. (Cited on page 5.)

[Harrold et al., 1998] Harrold, M. J., Rothermel, G., Wu, R., and Yi, L. (1998). An empirical investigation of program spectra. In *Proceedings of the 1998 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '98, pages 83–90, New York, NY, USA. ACM. (Cited on pages 41 and 49.)

[Heshmati et al., 2016] Heshmati, S., Kokkinogenis, Z., Rossetti, R., Carravilla, M., and Oliveira, J. (2016). An agent-based approach to schedule crane operations in rail-rail transshipment terminals. In Fonseca, R. J., Weber, G.-W., and Telhada, J., editors, *Computational Management Science*, volume 682 of *Lecture Notes in Economics and Mathematical Systems*, pages 91–97. Springer International Publishing. (Cited on page 86.)

[Hofer et al., 2015] Hofer, B., Perez, A., Abreu, R., and Wotawa, F. (2015). On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, 22(1):47–74. (Cited on pages 5, 7, 59, 66, 78, 82 and 83.)

[Hofer et al., 2013] Hofer, B., Riboira, A., Wotawa, F., Abreu, R., and Getzner, E. (2013). On the empirical evaluation of fault localization techniques for spreadsheets. In Cortellessa, V. and Varró, D., editors, *Fundamental Approaches to Software Engineering*, volume 7793 of *Lecture Notes in Computer Science*, pages 68–82. Springer-Verlag. (Cited on page 39.)

[Horling et al., 2001] Horling, B., Benyo, B., and Lesser, V. (2001). Using self-diagnosis to adapt organizational structures. *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536. (Cited on page 4.)

[Horling and Lesser, 2004] Horling, B. and Lesser, V. (2004). A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev.*, 19(4):281–316. (Cited on pages 31 and 61.)

[Horling et al., 2000] Horling, B., Lesser, V., Vincent, R., Bazzan, A., and Xuan, P. (2000). Diagnosis as an integral part of multi-agent adaptability. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 211 –219 vol.2. (Cited on pages 23, 30, 32, 34, 35 and 37.)

[Huang et al., 2014] Huang, Z., Alexander, R., and Clark, J. (2014). Mutation testing for jason agents. In Dalpiaz, F., Dix, J., and van Riemsdijk, M., editors, *Engineering Multi-Agent Systems*, volume 8758 of *Lecture Notes in Computer Science*, pages 309–327. Springer-Verlag. (Cited on page 63.)

[Huberman and Hogg, 1996] Huberman, B. A. and Hogg, T. (1996). Communities of practice: performance and evolution. *Computational and Mathmatical Organization Theory*, 1(1):73–92. (Cited on page 14.)

[Hubner and Sichman, 2003] Hubner, J. F. and Sichman, J. S. (2003). *SACI Programming Guide*. University of São Paulo. (Cited on page 87.)

[Isern et al., 2010] Isern, D., Sánchez, D., and Moreno, A. (2010). Agents applied in health care: A review. *International Journal of Medical Informatics*, 79(3):145 – 166. (Cited on page 1.)

[Isong and Bekele, 2013] Isong, B. E. and Bekele, E. (2013). A systematic review of fault tolerance in mobile agents. *American Journal of Software Engineering and Applications*, 2(5):111–124. (Cited on page 12.)

[Issicaba et al., 2012] Issicaba, D., Rosa, M. A., Franchin, W., and Lopes, J. A. P. (2012). Practical applications of agent-based technology. chapter Agent-Based System Applied to Smart Distribution Grid Operation, pages 1–20. InTech. (Cited on page 86.)

[Iyer et al., 1997] Iyer, R. K., Kalbarczyk, Z. T., and Bagchi, S. (1997). Chameleon: adaptive fault tolerance using reliable, mobile agents. In *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pages 61 –62. (Cited on page 26.)

[Jennings and Wooldridge, 1995] Jennings, N. R. and Wooldridge, M. J. (1995). Applying agent technology. *International Journal of Applied Artificial Intelligence*, 9(4):351–369. (Cited on page 11.)

[Jonge et al., 2009] Jonge, F., Roos, N., and Witteveen, C. (2009). Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems*, 18(2):267–294. (Cited on pages 4 and 24.)

[Kalbarczyk et al., 1999] Kalbarczyk, Z. T., Iyer, R. K., Bagchi, S., and Whisnant, K. (1999). Chameleon: a software infrastructure for adaptive fault tolerance. *Parallel and Distributed Systems, IEEE Transactions on*, 10(6):560 –579. (Cited on page 26.)

[Kalbarczyk et al., 2005] Kalbarczyk, Z. T., Iyer, R. K., and Wang, L. (2005). Application fault tolerance with armor middleware. *Internet Computing, IEEE*, 9(2):28–37. (Cited on pages 26, 29, 30, 32, 35, 36 and 37.)

[Kalech, 2012] Kalech, M. (2012). Diagnosis of coordination failures: a matrix-based approach. *Autonomous Agents and Multi-Agent Systems*, 24(1):69–103. (Cited on pages 4, 23, 30, 32, 35, 37 and 87.)

[Kalech and Kaminka, 2007] Kalech, M. and Kaminka, G. A. (2007). On the design of coordination diagnosis algorithms for teams of situated agents. *Artificial Intelligence*, 171:491 – 513. (Cited on pages 4 and 23.)

[Kalech and Kaminka, 2011] Kalech, M. and Kaminka, G. A. (2011). Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, 27(3):393–421. (Cited on pages 23, 51 and 57.)

[Kalech et al., 2007] Kalech, M., Lindner, M., and Kaminka, G. A. (2007). Matrix-based representation for coordination fault detection: a formal approach. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS '07, pages 162:1–162:8, New York, NY, USA. ACM. (Cited on pages 4 and 23.)

[Kaminka et al., 2002] Kaminka, G. A., Pynadath, D. V., and Tambe, M. (2002). Monitoring teams by overhearing: a multi-agent plan-recognition approach. *J. Artif. Int. Res.*, 17:83–135. (Cited on pages 4, 23, 30, 32, 35 and 37.)

[Kaminka and Tambe, 1998] Kaminka, G. A. and Tambe, M. (1998). What is wrong with us? improving robustness through social diagnosis. pages 97–104, Madison, WI, USA. AAAI. (Cited on pages 4, 23 and 33.)

[Kaminka and Tambe, 2000] Kaminka, G. A. and Tambe, M. (2000). Robust agent teams via socially-attentive monitoring. *J. Artif. Int. Res.*, 12:105–147. (Cited on pages 4 and 23.)

[Keil et al., 1995] Keil, M., Beranek, P. M., and Konsynski, B. R. (1995). Usefulness and ease of use: field study evidence regarding task considerations. *Decision Support Systems*, 13(1):75 – 91. User interfaces. (Cited on page 28.)

[Khalastchi et al., 2015] Khalastchi, E., Kalech, M., Kaminka, G., and Lin, R. (2015). Online data-driven anomaly detection in autonomous robots. *Knowledge and Information Systems*, 43(3):657–688. (Cited on pages 51 and 85.)

[Klein and Dellarocas, 1999] Klein, M. and Dellarocas, C. (1999). Exception handling in agent systems. In *Proceedings of the third annual conference on Autonomous Agents*, AGENTS '99, pages 62–68, New York, NY, USA. ACM. (Cited on pages 25 and 26.)

[Klein et al., 2003] Klein, M., Rodriguez-Aguilar, J.-A., and Dellarocas, C. (2003). Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *Autonomous Agents and Multi-Agent Systems*, 7:179–189. (Cited on pages 14, 26, 30, 32, 35, 37 and 48.)

[Knublauch, 2002] Knublauch, H. (2002). Extreme programming of multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, pages 704–711, New York, NY, USA. ACM. (Cited on pages 17, 30, 31, 32, 35 and 37.)

[Koca et al., 2013] Koca, F., Sozer, H., and Abreu, R. (2013). Spectrum-based fault localization for diagnosing concurrency faults. In Yenigun, H., Yilmaz, C., and Ulrich, A., editors, *Testing Software and Systems*, volume 8254 of *Lecture Notes in Computer Science*, pages 239–254. Springer-Verlag. (Cited on pages 5 and 82.)

[Koestler, 1967] Koestler, A. (1967). *The Ghost in the Machine*. Arkana Publishing, 1st edition. (Cited on page 13.)

[Kokkinogenis et al., 2011] Kokkinogenis, Z., Passos, L. S., Rossetti, R. J. F., and Gabriel, J. (2011). Towards the next-generation traffic simulation tools: a first evaluation. In *Proceedings of the 6th Doctoral Symposium in Informatics Engineering (DSIE 2011)*. (Cited on page 92.)

[Kolodner, 1992] Kolodner, J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34. (Cited on page 87.)

[Kumar and Cohen, 2000] Kumar, S. and Cohen, P. R. (2000). Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the fourth international conference on Autonomous agents*, AGENTS '00, pages 459–466, New York, NY, USA. ACM. (Cited on page 26.)

[Kumar et al., 2000] Kumar, S., Cohen, P. R., and Levesque, H. J. (2000). The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 159–, Washington, DC, USA. IEEE Computer Society. (Cited on pages 26, 30, 32, 35 and 37.)

[Lam and Barber, 2004] Lam, D. N. and Barber, K. S. (2004). Verifying and explaining agent behavior in an implemented agent system. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 1226–1227. (Cited on page 18.)

[Lam and Barber, 2005a] Lam, D. N. and Barber, K. S. (2005a). Comprehending agent software. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 586–593, New York, NY, USA. ACM. (Cited on pages 5, 18, 30, 32, 35 and 37.)

[Lam and Barber, 2005b] Lam, D. N. and Barber, K. S. (2005b). Debugging agent behavior in an implemented agent system. In Bordini, R. H., Dastani, M., Dix, J., and El Fallah Seghrouchni, A., editors, *Programming Multi-Agent Systems*, volume 3346 of *Lecture Notes in Computer Science*, pages 104–125. Springer-Verlag. (Cited on pages 5 and 18.)

[Laprie et al., 1992] Laprie, J.-C., Avižienis, A., and Kopetz, H., editors (1992). *Dependability: Basic Concepts and Terminology - In English, French,German, Italian and Japanese*. Springer-Verlag, Secaucus, NJ, USA. (Cited on page 13.)

[Le et al., 2013] Le, T.-D. B., Thung, F., and Lo, D. (2013). Theory and practice, do they match? a case with spectrum-based fault localization. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 380–383. (Cited on pages 5, 7, 39, 59, 78, 82 and 83.)

[Lee and Anderson, 1990] Lee, P. A. and Anderson, T. (1990). *Fault Tolerance: Principles and Practice*. Springer-Verlag, Secaucus, NJ, USA, 2nd edition. (Cited on pages 13 and 22.)

[Leitão et al., 2012] Leitão, P., Mařík, V., and Vrba, P. (2012). Past, present, and future of industrial agent applications. *Industrial Informatics, IEEE Transactions on*, PP(99):1–1. (Cited on pages 1, 13 and 63.)

[Leitão and Vrba, 2011] Leitão, P. and Vrba, P. (2011). Recent developments and future trends of industrial agents. In Mařík, V., Vrba, P., and Leitão, P., editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 6867 of *Lecture Notes in Computer Science*, pages 15–28. Springer-Verlag. (Cited on page 1.)

[Lettmann et al., 2011] Lettmann, T., Baumann, M., Eberling, M., and Kemmerich, T. (2011). Modeling agents and agent systems. In Nguyen, N. T., editor, *Transactions on Computational Collective Intelligence V*, volume 6910 of *Lecture Notes in Computer Science*, pages 157–181. Springer-Verlag. (Cited on page 45.)

[Liskov and Snyder, 1979] Liskov, B. H. and Snyder, A. (1979). Exception handling in clu. *IEEE Transactions on Software Engineering*, 5(6):546–558. (Cited on page 13.)

[Lomuscio and Michaliszyn, 2014] Lomuscio, A. and Michaliszyn, J. (2014). Decidability of model checking multi-agent systems against some ehs specifications. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI14)*, pages 543–548, Prague, Czech Republic. IOS Press. (Cited on page 4.)

[Lomuscio and Michaliszyn, 2015] Lomuscio, A. and Michaliszyn, J. (2015). Verifying multi-agent systems by model checking three-valued abstractions. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, AAMAS '15, Istanbul, Turkey. IFAAMAS Press. (Cited on page 4.)

[Lomuscio and Paquet, 2015] Lomuscio, A. and Paquet, H. (2015). Verification of multi-agent systems via sdd-based model checking (extended abstract). In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, AAMAS '15, pages 1713–1714, Istanbul, Turkey. IFAAMAS Press. (Cited on page 4.)

[Lomuscio et al., 2009] Lomuscio, A., Qu, H., and Raimondi, F. (2009). Mcmas: A model checker for the verification of multi-agent systems. In Bouajjani, A. and Maler, O., editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 682–688. Springer-Verlag. (Cited on page 4.)

[Lucas, 1998] Lucas, P. J. (1998). Analysis of notions of diagnosis. *Artificial Intelligence*, 105:295 – 343. (Cited on page 3.)

[Lucia et al., 2014] Lucia, Lo, D., Jiang, L., Thung, F., and Budi, A. (2014). Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 26(2):172–219. (Cited on page 66.)

[Maffioletti et al., 2013] Maffioletti, F., Reffato, R., Farinelli, A., Kleiner, A., Ramchurn, S., and Shi, B. (2013). Rmasbench: A benchmarking system for multi-agent coordination in urban search and rescue. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1383–1384, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on page 88.)

[Mallya and Singh, 2005a] Mallya, A. and Singh, M. (2005a). A semantic approach for designing commitment protocols. In van Eijk, R., Huget, M.-P., and Dignum, F., editors, *Agent Communication*, volume 3396 of *Lecture Notes in Computer Science*, pages 33–49. Springer-Verlag. (Cited on pages 25, 30, 32, 34, 35 and 37.)

[Mallya and Singh, 2005b] Mallya, A. U. and Singh, M. P. (2005b). Modeling exceptions via commitment protocols. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 122–129, New York, NY, USA. ACM. (Cited on pages 15 and 25.)

[Manvi and Venkataram, 2004] Manvi, S. and Venkataram, P. (2004). Applications of agent technology in communications: a review. *Computer Communications*, 27(15):1493 – 1508. (Cited on page 1.)

[Marin et al., 2007] Marin, O., Bertier, M., Sens, P., Guessoum, Z., and Briot, J.-P. (2007). Darx - a self-healing framework for agents. In Kordon, F. and Sztipanovits, J., editors, *Reliable Systems on Unreliable Networked Platforms*, volume 4322 of *Lecture Notes in Computer Science*, pages 88–105. Springer-Verlag. (Cited on page 27.)

[Mayer and Stumptner, 2007] Mayer, W. and Stumptner, M. (2007). Model-based debugging - state of the art and future challenges. *Electronic Notes in Theoretical Computer Science*, 174(4):61–82. Proceedings of the Workshop on Verification and Debugging (V&D 2006). (Cited on page 5.)

[Mcburney and Omicini, 2008] Mcburney, P. and Omicini, A. (2008). Editorial: Special issue on foundations, advanced topics and industrial perspectives of multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17:367–371. (Cited on page 2.)

[McQuitty, 1966] McQuitty, L. L. (1966). Similarity analysis by reciprocal pairs for discrete and continuous data. *Educational and Psychological Measurement*, 26(4):825–831. (Cited on page 69.)

[Mellouli et al., 2004] Mellouli, S., Moulin, B., and Mineau, G. (2004). Laying down the foundations of an agent modelling methodology for fault-tolerant multi-agent systems. In Omicini, A., Petta, P., and Pitt, J., editors, *Engineering Societies in the Agents World IV*, volume 3071 of *Lecture Notes in Computer Science*, pages 519–519. Springer-Verlag. (Cited on page 15.)

[Micalizio, 2009] Micalizio, R. (2009). A distributed control loop for autonomous recovery in a multi-agent plan. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1760–1765, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. (Cited on pages 4 and 24.)

[Micalizio, 2013] Micalizio, R. (2013). Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, 29(2):233–280. (Cited on pages 4, 24 and 87.)

[Micalizio and Torasso, 2014] Micalizio, R. and Torasso, P. (2014). Cooperative monitoring to diagnose multiagent plans. *Journal of Artificial Intelligence Research*, 51(1):1–70. (Cited on pages 24, 30, 32, 35 and 37.)

[Michel et al., 2009] Michel, F., Ferber, J., and Drogoul, A. (2009). Multi-agent systems and simulation: a survey from the agents community's perspective. In Danny Weyns, A. U., editor, *Multi-Agent Systems: Simulation and Applications*, Computational Analysis, Synthesis, and Design of Dynamic Systems, page 47. CRC Press - Taylor & Francis. (Cited on pages 12 and 20.)

[Miller and Tripathi, 2004] Miller, R. and Tripathi, A. (2004). The guardian model and primitives for exception handling in distributed systems. *Software Engineering, IEEE Transactions on*, 30(12):1008 – 1022. (Cited on pages 4, 26, 30, 32, 35 and 37.)

[Mishra, 2001] Mishra, S. (2001). Agent fault tolerance using group communication. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada, USA. CSREA Publishing. (Cited on page 26.)

[Mishra and Huang, 2000] Mishra, S. and Huang, Y. (2000). Fault tolerance in agent-based computing systems. In *In Proceedings of the 13th ISCA International Conference on Parallel and Distributed Computing Systems*, Las Vegas, NV, USA. (Cited on page 26.)

[Mishra and Xie, 2003] Mishra, S. and Xie, P. (2003). Interagent communication and synchronization support in the daagent mobile agent-based computing system. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):290–306. (Cited on pages 26, 30, 32, 35 and 37.)

[Moreno et al., 2009] Moreno, M., Pavón, J., and Rosete, A. (2009). Testing in agent oriented methodologies. In Omatu, S., Rocha, M. P., Bravo, J., Fernández, F., Corchado, E., Bustillo, A., and Corchado, J. M., editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 138–145. Springer-Verlag. (Cited on page 16.)

[Nguyen et al., 2012] Nguyen, C. D., Miles, S., Perini, A., Tonella, P., Harman, M., and Luck, M. (2012). Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, 25(2):260–283. (Cited on pages 4, 7, 20, 30, 31, 32, 35, 36, 37 and 88.)

[Nguyen et al., 2011] Nguyen, C. D., Perini, A., Bernon, C., Pavón, J., and Thangarajah, J. (2011). Testing in multi-agent systems. In Gleizes, M.-P. and Gomez-Sanz, J., editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 180–190. Springer-Verlag. (Cited on pages 12, 28 and 39.)

[Nguyen et al., 2008] Nguyen, C. D., Perini, A., and Tonella, P. (2008). ecat: A tool for automating test cases generation and execution in testing multi-agent systems (demo paper). In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Demo Papers*, AAMAS '08, pages 1669–1670, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems. (Cited on page 20.)

[Nguyen et al., 2009] Nguyen, C. D., Perini, A., and Tonella, P. (2009). Experimental evaluation of ontology-based test generation for multi-agent systems. In Luck, M. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 187–198. Springer-Verlag. (Cited on page 20.)

[Nguyen et al., 2010] Nguyen, C. D., Perini, A., and Tonella, P. (2010). Goal-oriented testing for mass. *Int. J. Agent-Oriented Softw. Eng.*, 4(1):79–109. (Cited on page 20.)

[Núñez et al., 2005] Núñez, M., Rodríguez, I., and Rubio, F. (2005). Specification and testing of autonomous agents in e-commerce systems: Research articles. *Softw. Test. Verif. Reliab.*, 15(4):211–233. (Cited on page 18.)

[Padgham et al., 2007] Padgham, L., Thangarajah, J., and Winikoff, M. (2007). Auml protocols and code generation in the prometheus design tool. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 270:1–270:2, New York, NY, USA. ACM. (Cited on page 20.)

[Padgham and Winikoff, 2004] Padgham, L. and Winikoff, M. (2004). *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons, Inc., New York, NY, USA. (Cited on page 20.)

[Padgham et al., 2005] Padgham, L., Winikoff, M., and Poutakidis, D. (2005). Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173 – 190. Agent-oriented Software Development. (Cited on page 20.)

[Padgham et al., 2013] Padgham, L., Zhang, Z., Thangarajah, J., and Miller, T. (2013). Model-based test oracle generation for automated unit testing of agent systems. *Software Engineering, IEEE Transactions on*, 39(9):1230–1244. (Cited on pages 20, 30, 31, 32, 35, 37 and 88.)

[Pardo et al., 2010] Pardo, J. J., Núñez, M., and Ruiz, M. C. (2010). Specification and testing of e-commerce agents described by using uioltss. In Hatcliff, J. and Zucca, E., editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 78–86. Springer-Verlag. (Cited on pages 18, 30, 32, 35 and 37.)

[Parsons and Wooldridge, 2002] Parsons, S. and Wooldridge, M. (2002). Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 5(3):243–254. (Cited on page 1.)

[Parunak, 1996] Parunak, H. V. D. (1996). *Applications of distributed artificial intelligence in industry*, pages 139–164. John Wiley & Sons, Inc., New York, NY, USA. (Cited on pages 1 and 11.)

[Parunak, 2000] Parunak, H. V. D. (2000). A practitioners' review of industrial agent applications. *Autonomous Agents and Multi-Agent Systems*, 3(4):389–407. (Cited on page 1.)

[Passos, 2010] Passos, L. (2010). Evaluating agent-based methodologies with focus on transportation systems. In *Proceedings of the 5th Doctoral Symposium in Informatics Engineering (DSIE 2010)*. (Cited on page 93.)

[Passos et al., 2013a] Passos, L., Kokkinogenis, Z., Rossetti, R., and Gabriel, J. (2013a). Multi-resolution simulation of taxi services on airport terminal's curbside. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, pages 2361–2366. (Cited on page 92.)

[Passos et al., 2011a] Passos, L., Kokkinogenis, Z., and Rossetti, R. J. F. (2011a). Towards the next-generation traffic simulation tools: a first appraisal. In *Proceedings of the 3th Workshop on Intelligent Systems and Applications (WISA 2011), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2011)*. (Cited on page 92.)

[Passos and Rossetti, 2009] Passos, L. and Rossetti, R. J. F. (2009). Intelligent transportation systems: a ubiquitous perspective. In *New Trends in Artificial Intelligence collocated in the Proceedings of the 14th Portuguese Conference on Artificial Intelligence (EPIA)*. (Cited on page 93.)

[Passos and Rossetti, 2010a] Passos, L. and Rossetti, R. J. F. (2010a). Traffic light control using reactive agents. In *Proceedings of the 2nd Workshop on Intelligent Systems and Applications (WISA 2010), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2010)*. (Cited on page 93.)

[Passos et al., 2011b] Passos, L., Rossetti, R. J. F., and Reis, L. P. (2011b). Evaluation of taxi services on airport terminal's curbside for picking up passengers. In *Proceedings of the 3th Workshop on Intelligent Systems and Applications (WISA 2011), collocated with the 5th Iberian Conference on Information Systems and Technologies (CISTI 2011)*. (Cited on page 92.)

[Passos et al., 2014] Passos, L. S., Abreu, R., and Rossetti, R. J. F. (2014). Sensitivity analysis of spectrum-based fault localisation for multi-agent systems. In *Proceedings of the 25th International Workshop on Principles of Diagnosis (DX'14)*. (Cited on page 9.)

[Passos et al., 2015a] Passos, L. S., Abreu, R., and Rossetti, R. J. F. (2015a). Spectrum-based fault localisation for multi-agent systems. In Yang, Q. and Wooldridge, M., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1134–1140. AAAI Press. (Cited on page 9.)

[Passos et al., 2013b] Passos, L. S., Rossetti, R., and Gabriel, J. (2013b). Diagnosis of unwanted behaviours in multi-agent systems. In *Proceedings of the 11th European Workshop on Multi-Agent Systems*. (Cited on page 9.)

[Passos et al., 2015b] Passos, L. S., Rossetti, R. J., and Gabriel, J. (2015b). An agent methodology for processes, the environment, and services. In Rossetti, R. J. and Ronghui, L., editors, *Advances in Artificial Transportation Systems and Simulation*, pages 37 – 53. Academic Press, Boston. (Cited on page 92.)

[Passos and Rossetti, 2010b] Passos, L. S. and Rossetti, R. J. F. (2010b). Traffic light control using reactive agents. In *Information Systems and Technologies (CISTI), 2010 5th Iberian Conference on*, pages 1–6. (Cited on page 63.)

[Passos et al., 2015c] Passos, L. S., Rossetti, R. J. F., and Gabriel, J. (2015c). Advances in fault-tolerant multi-agent systems. In Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Encyclopedia of Information Science and Technology, Third Edition*, pages 7006–7017. IGI Global. (Cited on pages 8 and 12.)

[Passos et al., 2010] Passos, L. S., Rossetti, R. J. F., and Oliveira, E. C. (2010). Ambient-centred intelligent traffic control and management. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 224–229. (Cited on page 93.)

[Patterson et al., 2002] Patterson, D., Brown, A., Broadwell, P., Candea, G., Chen, M., Cutler, J., Enriquez, P., Fox, A., Kiciman, E., Merzbacher, M., Oppenheimer, D., Sastry, N., Tetzlaff, W., Traupman, J., and Treuhaft, N. (2002). Recovery oriented computing (roc): Motivation, definition, techniques, and case studies. Technical Report UCB/CSD-02-1175, EECS Department, University of California, Berkeley. (Cited on page 3.)

[Pearson, 1895] Pearson, K. (1895). Note on regression and inheritance in the case of two parents. *Proceedings of the Royal Society of London*, 58:240–242. (Cited on page 69.)

[Pěchouček and Mařík, 2008] Pěchouček, M. and Mařík, V. (2008). Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17:397–431. (Cited on pages 1, 2 and 11.)

[Platon et al., 2007] Platon, E., Sabouret, N., and Honiden, S. (2007). A definition of exceptions in agent-oriented computing. In *Proceedings of the 7th international conference on Engineering societies in the agents world VII*, ESAW'06, pages 161–174, Berlin, Heidelberg. Springer-Verlag. (Cited on pages 14 and 48.)

[Platon et al., 2008] Platon, E., Sabouret, N., and Honiden, S. (2008). An architecture for exception management in multiagent systems. *Int. J. Agent-Oriented Softw. Eng.*, 2:267–289. (Cited on pages 16, 25, 30, 32, 34, 35 and 37.)

[Posadas et al., 2008] Posadas, J. L., Poza, J. L., Simó, J. E., Benet, G., and Blanes, F. (2008). Agent-based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*, 21(6):805 – 823. (Cited on pages 60 and 63.)

[Potiron et al., 2008] Potiron, K., Taillibert, P., and Fallah Seghrouchni, A. (2008). Languages, methodologies and development tools for multi-agent systems. chapter A Step Towards Fault Tolerance for Multi-Agent Systems, pages 156–172. Springer-Verlag, Berlin, Heidelberg. (Cited on page 15.)

[Poutakidis et al., 2003] Poutakidis, D., Padgham, L., and Winikoff, M. (2003). An exploration of bugs and debugging in multi-agent systems. In Zhong, N., Raś, Z. W., Tsumoto, S., and Suzuki, E., editors, *Foundations of Intelligent Systems*, volume 2871 of *Lecture Notes in Computer Science*, pages 628–632. Springer-Verlag. (Cited on page 20.)

[Poutakidis et al., 2009] Poutakidis, D., Winikoff, M., Padgham, L., and Zhang, Z. (2009). Debugging and testing of multi-agent systems using design artefacts. In El Fallah Seghrouchni, A., Dix, J., Dastani, M., and Bordini, R. H., editors, *Multi-Agent Programming:*, pages 215–258. Springer-Verlag. (Cited on page 20.)

[Powell et al., 1988] Powell, D., Bonn, G., Seaton, D., Veríssimo, P., and Waeselynck, F. (1988). The delta-4 approach to dependability in open distributed computing systems. In *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, pages 246 –251. (Cited on page 15.)

[Qin et al., 2014] Qin, L., He, X., and Zhou, D. H. (2014). A survey of fault diagnosis for swarm systems. *Systems Science & Control Engineering*, 2(1):13–23. (Cited on page 16.)

[Qu et al., 2005] Qu, W., Shen, H., and Defago, X. (2005). A survey of mobile agent-based fault-tolerant technology. In *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, pages 446–450. (Cited on page 12.)

[Rafael H. Bordini, 2007] Rafael H. Bordini, Jomi Fred Hubner, M. W. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, Ltd. (Cited on page 61.)

[Rao, 1996] Rao, A. S. (1996). Agentspeak(l): Bdi agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer-Verlag. (Cited on page 61.)

[Reeven, 2011] Reeven, B. J. (2011). Model-based diagnosis in industrial context. Master's thesis, Rijksuniversiteit Groningen, The Netherlands. (Cited on page 5.)

[Reiter, 1987] Reiter, R. (1987). A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95. (Cited on page 48.)

[Reps et al., 1997] Reps, T., Ball, T., Das, M., and Larus, J. (1997). The use of program profiling for software maintenance with applications to the year 2000 problem. In Jazayeri, M. and Schauer, H., editors, *Software Engineering ESEC/FSE'97*, volume 1301 of *Lecture Notes in Computer Science*, pages 432–449. Springer-Verlag. (Cited on pages 5, 39 and 41.)

[Rodrigues et al., 2013] Rodrigues, N., Leitão, P., Foehr, M., Turrin, C., Pagani, A., and Decesari, R. (2013). Adaptation of functional inspection test plan in a production line using a multi-agent system. In *Industrial Electronics (ISIE), 2013 IEEE International Symposium on*, pages 1–6. (Cited on page 86.)

[Roos et al., 2004] Roos, N., ten Teije, A., and Witteveen, C. (2004). Reaching diagnostic agreement in multi-agent diagnosis. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 1256 –1257. (Cited on page 4.)

[Roos and Witteveen, 2009] Roos, N. and Witteveen, C. (2009). Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19(1):30–52. (Cited on pages 4 and 24.)

[Rossetti et al., 2008] Rossetti, R., Ferreira, P., Braga, R., and Oliveira, E. (2008). Towards an artificial traffic control system. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 14–19. (Cited on page 86.)

[Rossetti et al., 2002] Rossetti, R. J., Bordini, R. H., Bazzan, A. L., Bampi, S., Liu, R., and Vliet, D. V. (2002). Using BDI agents to improve driver modelling in a commuter scenario. *Transportation Research Part C: Emerging Technologies*, 10(5-6):373 – 398. (Cited on page 61.)

[Rossetti and Liu, 2005a] Rossetti, R. J. and Liu, R. (2005a). A dynamic network simulation model based on multi-agent systems. In Klugl, F., Bazzan, A., and Ossowski, S., editors, *Applications of Agent Technology in Traffic and Transportation*, Whitestein Series in Software Agent Technologies, pages 181–192. Birkauser Basel. (Cited on page 61.)

[Rossetti and Liu, 2005b] Rossetti, R. J. F. and Liu, R. (2005b). An agent-based approach to assess drivers' interaction with pre-trip information systems. *Journal of Intelligent Transportation Systems*, 9(1):1–10. (Cited on page 61.)

[Rossetti and Liu, 2005c] Rossetti, R. J. F. and Liu, R. (2005c). Progress in activity-based analysis. chapter Activity-Based Analysis of Travel Demand Using Cognitive Agents, pages 139–160. National Academy of Sciences. (Cited on page 61.)

[Rossetti et al., 2007] Rossetti, R. J. F., Oliveira, E. C., and Bazzan, A. L. C. (2007). Towards a specification of a framework for sustainable transportation analysis. In *Proceedings of the 13th Portuguese Conference on Artificial Intelligence (EPIA)*. (Cited on page 86.)

[Rustogi et al., 1999] Rustogi, S. K., Wan, F., Xing, J., and Singh, M. P. (1999). Handling semantic exceptions in the large: A multiagent approach. Technical report, Raleigh, NC, USA. (Cited on pages 24, 30, 32, 34, 35 and 37.)

[Savelsbergh and Sol, 1995] Savelsbergh, M. W. P. and Sol, M. (1995). The general pickup and delivery problem. *Transportation Science*, 29(1):17–29. (Cited on pages 7 and 60.)

[Serrano and Botia, 2009] Serrano, E. and Botia, J. A. (2009). Infrastructure for forensic analysis of multi-agent systems. In Hindriks, K. V., Pokahr, A., and Sardina, S., editors, *Programming Multi-Agent Systems*, volume 5442 of *Lecture Notes in Computer Science*, pages 168–183. Springer-Verlag. (Cited on page 19.)

[Serrano et al., 2009] Serrano, E., Gómez-Sanz, J. J., Botía, J. A., and Pavón, J. (2009). Intelligent data analysis applied to debug complex software systems. *Neurocomputing*, 72(13–15):2785–2795. Hybrid Learning Machines (HAIS 2007) / Recent

Developments in Natural Computation (ICNC 2007). (Cited on page 19.)

[Serrano et al., 2012] Serrano, E., Munoz, A., and Botía, J. (2012). An approach to debug interactions in multi-agent system software tests. *Information Sciences*, 205(0):38–57. (Cited on pages 4, 19, 29, 30, 32, 35 and 37.)

[Shah et al., 2009] Shah, N., Iqbal, R., James, A., and Iqbal, K. (2009). Exception representation and management in open multi-agent systems. *Information Sciences*, 179(15):2555 – 2561. Including Special Issue on Computer-Supported Cooperative WorkTechniques and ApplicationsThe 11th Edition of the International Conference on CSCW in Design. (Cited on page 4.)

[Shimodaira, 2004] Shimodaira, H. (2004). Approximately unbiased tests of regions using multistep-multiscale bootstrap resampling. *The Annals of Statistics*, 32(6):2616–2641. (Cited on page 69.)

[Sierhuis et al., 2009] Sierhuis, M., Clancey, W. J., van Hoof, R. J., Seah, C. H., Scott, M. S., Nado, R. A., Blumenberg, S. F., Shafto, M. G., Anderson, B. L., Bruins, A. C., Buckley, C. B., Diegelman, T. E., Hall, T. A., Hood, D., Reynolds, F. F., Toschlog, J. R., and Tucker, T. (2009). Nasa's oca mirroring system - an application of multiagent systems in mission control. *8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 85–92. (Cited on page 1.)

[Sierra et al., 2004] Sierra, C., Rodríguez-Aguilar, J. A., Noriega, P., Esteva, M., and Arcos, J. L. (2004). Engineering multi-agent systems as electronic institutions. *UPGRADE The European Journal for the Informatics Professional*, V(4):33–39. (Cited on pages 21, 30, 31, 32, 35 and 37.)

[Smith and Korsmeyer, 2012] Smith, E. E. and Korsmeyer, D. J. (2012). Intelligent systems technologies for ops. In Wickler, M., editor, *12th International Conference on Space Operations (SpaceOps 2012)*, Stockholm. NASA Ames Research Center. (Cited on page 1.)

[Souchon et al., 2003] Souchon, F., Dony, C., Urtado, C., and Vauttier, S. (2003). A proposition for exception handling in multi-agent systems. In *Proceedings of the 2nd internationaal workshop on Software Engineering for Large-Scale Multi-Agent Systems at ICSE'03*, pages 136–143. (Cited on page 24.)

[Souchon et al., 2004] Souchon, F., Dony, C., Urtado, C., and Vauttier, S. (2004). Improving exception handling in multi-agent systems. In Lucena, C., Garcia, A., Romanovsky, A., Castro, J., and Alencar, P., editors, *Software Engineering for Multi-Agent Systems II*, volume 2940 of *Lecture Notes in Computer Science*, pages 333–337. Springer-Verlag. (Cited on pages 14, 24, 29, 30, 32, 35 and 37.)

[Sözer et al., 2010] Sözer, H., Abreu, R., Aksit, M., and van Gemund, A. J. (2010). Increasing system availability with local recovery based on fault localization. In *Proceedings of the 10th International Conference on Quality Software, QSIC 2010*, pages 276–281. IEEE Computer Society. (Cited on page 3.)

[Steimann et al., 2013] Steimann, F., Frenkel, M., and Abreu, R. (2013). Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 314–324, New York, NY, USA. ACM. (Cited on page 65.)

[Stone and Veloso, 2000] Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383. (Cited on pages 11 and 61.)

[Struss and Dressler, 1992] Struss, P. and Dressler, O. (1992). Readings in model-based diagnosis. chapter &Ldquo;Physical Negation&Rdquo;&Mdash;Integrating Fault Models into the General Diagnostic Engine, pages 153–158. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (Cited on page 5.)

[Sudeikat and Renz, 2009] Sudeikat, J. and Renz, W. (2009). A systemic approach to the validation of self-organizing dynamics within mas. In Luck, M. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 31–45. Springer-Verlag. (Cited on page 22.)

[Sudeikat and Renz, 2011] Sudeikat, J. and Renz, W. (2011). Qualitative modeling of mas dynamics. In Gleizes, M.-P. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 80–93. Springer-Verlag. (Cited on pages 22, 30, 32, 35 and 37.)

[Tacla and Barthès, 2003] Tacla, C. A. and Barthès, J.-P. (2003). A multi-agent system for acquiring and sharing lessons learned. *Computers in Industry*, 52(1):5 – 16. <ce:title>Knowledge Sharing in Collaborative Design Environments</ce:title>. (Cited on page 1.)

[Timóteo et al., 2012] Timóteo, I., Araújo, M. R., Rossetti, R. J., and Oliveira, E. C. (2012). Using trasmapi for the assessment of multi-agent traffic management solutions. *Progress in Artificial Intelligence*, 1(2):157–164. (Cited on page 86.)

[Tiryaki et al., 2007] Tiryaki, A., Öztuna, S., Dikenelli, O., and Erdur, R. (2007). Sunit: A unit testing framework for test driven development of multi-agent systems. In Padgham, L. and Zambonelli, F., editors, *Agent-Oriented Software Engineering VII*, volume 4405 of *Lecture Notes in Computer Science*, pages 156–173. Springer-Verlag. (Cited on page 19.)

[Tripathi and Miller, 2001] Tripathi, A. and Miller, R. (2001). Exception handling in agent-oriented systems. In Romanovsky, A., Dony, C., Knudsen, J. r., and Tripathi, A., editors, *Advances in Exception Handling Techniques*, volume 2022 of *Lecture Notes in Computer Science*, pages 128–146. Springer-Verlag. (Cited on pages 14 and 26.)

[Čermák et al., 2015] Čermák, P., Lomuscio, A., and Murano, A. (2015). Verifying and synthesising multi-agent systems against one-goal strategy logic specifications. (Cited on page 4.)

[Veloso et al., 2014a] Veloso, R., Kokkinogenis, Z., Passos, L., Oliveira, G., Rossetti, R., and Gabriel, J. (2014a). A platform for the design, simulation and development of quadcopter multi-agent systems. In *Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on*, pages 1–6. (Cited on page 95.)

[Veloso et al., 2014b] Veloso, R., Oliveira, G., Passos, L., Kokkinogenis, Z., Rossetti, R., and Gabriel, J. (2014b). A symbiotic simulation platform for agent-based quadcopters. In *Information Systems and Technologies (CISTI), 2014 9th Iberian Conference on*, pages 1–6. (Cited on page 95.)

[Vieira-Marques et al., 2006] Vieira-Marques, P., Cruz-Correia, R., Robles, S., Cucurull, J., Navarro, G., and Marti, R. (2006). Secure integration of distributed medical data using mobile agents. *Intelligent Systems, IEEE*, 21(6):47–54. (Cited on page 1.)

[Vilarinho et al., 2015] Vilarinho, C., Tavares, J. P., and Rossetti, R. J. (2015). A conceptual mas model for real-time traffic control. In Pereira, F., Machado, P., Costa, E., and Cardoso, A., editors, *Progress in Artificial Intelligence*, volume 9273 of *Lecture Notes in Computer Science*, pages 157–168. Springer International Publishing. (Cited on page 86.)

[Wagner, 2000] Wagner, D. N. (2000). Liberal order for software agents? an economic analysis. *Journal of Artificial Societies and Social Simulation*, 3(1). (Cited on pages xiii, 15, 28, 36 and 37.)

[Winikoff, 2010] Winikoff, M. (2010). Assurance of agent systems: What role should formal verification play? In Dastani, M., Hindriks, K. V., and Meyer, J.-J. C., editors, *Specification and Verification of Multi-agent Systems*, pages 353–383. Springer-Verlag. (Cited on page 88.)

[Wittig, 1992] Wittig, T., editor (1992). *ARCHON: an architecture for multi-agent systems*. Ellis Horwood, Upper Saddle River, NJ, USA. (Cited on page 1.)

[Wong et al., 2012] Wong, W., Debroy, V., Li, Y., and Gao, R. (2012). Software fault localization using dstar (d*). In *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*, pages 21–30. (Cited on page 87.)

[Wong et al., 2010] Wong, W., Debroy, V., Surampudi, A., Kim, H., and Siok, M. (2010). Recent catastrophic accidents: Investigating how software was responsible. In *Secure Software Integration and Reliability Improvement (SSIRI), 2010 Fourth International Conference on*, pages 14–22. (Cited on pages 2 and 11.)

[Wooldridge, 2009] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition. (Cited on page 11.)

[Xu and Deters, 2004a] Xu, P. and Deters, R. (2004a). Mas & fault-management. In *Applications and the Internet, 2004. Proceedings. 2004 International Symposium on*, pages 283 – 286. (Cited on pages 14 and 24.)

[Xu and Deters, 2004b] Xu, P. and Deters, R. (2004b). Using event-streams for fault-management in mas. In *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004. (IAT 2004).*, pages 433–436. IEEE. (Cited on page 24.)

[Xu and Deters, 2005] Xu, P. and Deters, R. (2005). Fault-management for multi-agent systems. In *Applications and the Internet, 2005. Proceedings. The 2005 Symposium on*, pages 287 – 293. (Cited on pages 24, 28, 30, 32, 35 and 37.)

[Yoo et al., 2014] Yoo, S., Xie, X., Kuo, F.-C., Chen, T. Y., , and Harman, M. (2014). No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist. Technical Report Technical Report RN/14/14, University College London. (Cited on pages 7, 59 and 83.)

[Zamir et al., 2014] Zamir, T., Stern, R., and Kalech, M. (2014). Using model-based diagnosis to improve software testing. (Cited on page 48.)

[Zhang et al., 2009] Zhang, Z., Thangarajah, J., and Padgham, L. (2009). Model based testing for agent systems. In Filipe, J., Shishkov, B., Helfert, M., and Maciaszek, L. A., editors, *Software and Data Technologies*, volume 22 of *Communications in Computer and Information Science*, pages 399–413. Springer-Verlag. (Cited on page 20.)

[Zhang et al., 2011] Zhang, Z., Thangarajah, J., and Padgham, L. (2011). Automated testing for intelligent agent systems. In Gleizes, M.-P. and Gomez-Sanz, J. J., editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 66–79. Springer-Verlag. (Cited on page 20.)

[Zoeteweij et al., 2007] Zoeteweij, P., Abreu, R., Golsteijn, R., and van Gemund, A. J. (2007). Diagnosis of embedded software using program spectra. In *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, pages 213–220. (Cited on page 39.)